

Cours 10

Fonctions récursives
Graphisme

Vincent Lepetit

`vincent.lepetit@epfl.ch`

Fonctions récursives

Une fonction peut s'appeler elle-même

La fonction *factorielle* peut se définir ainsi:

factorielle(0) = 1

factorielle(n) = n × *factorielle*(n - 1) si n > 0

On peut utiliser cette définition pour écrire la fonction C `fact`:

```
int fact(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
```

...

```
cout << fact(5) << endl;
```

Les fonctions qui s'appellent elles-mêmes sont nommées **fonctions récursives**.

Fonctions récursives

Comment l'ordinateur exécute une fonction récursive ?

Tout se passe comme si à chaque fois que la fonction s'appelle elle-même, la fonction était dupliquée pour créer une nouvelle fonction, identique mais avec ses propres paramètres et variables locales.

Ce qui se passe en réalité est en fait assez proche: la fonction n'est pas vraiment dupliquée mais les paramètres et les variables locales le sont.

Les transparents suivants détaillent l'appel de la fonction `fact` quand on lui passe la valeur 3 en paramètre.

Pas-à-pas

```
int fact(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
```

...

 cout << fact(3) << endl;

Pas-à-pas

```
→ int fact(int n) n = 3  
  {  
    if (n == 0)  
      return 1;  
    else  
      return n * fact(n - 1);  
  }  
  
  ...  
  
  cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3  
{  
→ if (n == 0) La condition est fausse...  
    return 1;  
    else  
        return n * fact(n - 1);  
}
```

...

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3  
{  
    if (n == 0)  
        return 1;  
    else  
→ return n * fact(n - 1);  
}
```

...c'est donc cette instruction-ci
qui est exécutée

...

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        → return n * fact(n - 1);
}

...
```

A l'appel de `fact(n-1)`, tout se passe comme si une nouvelle fonction `fact` était créée, puis appelée en lui passant 2 ($=n-1$) en paramètre.

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

...

```
cout << fact(3) << endl;
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
```

La nouvelle fonction `fact` a elle-aussi un paramètre `n`, qui vaut 2 ici.

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

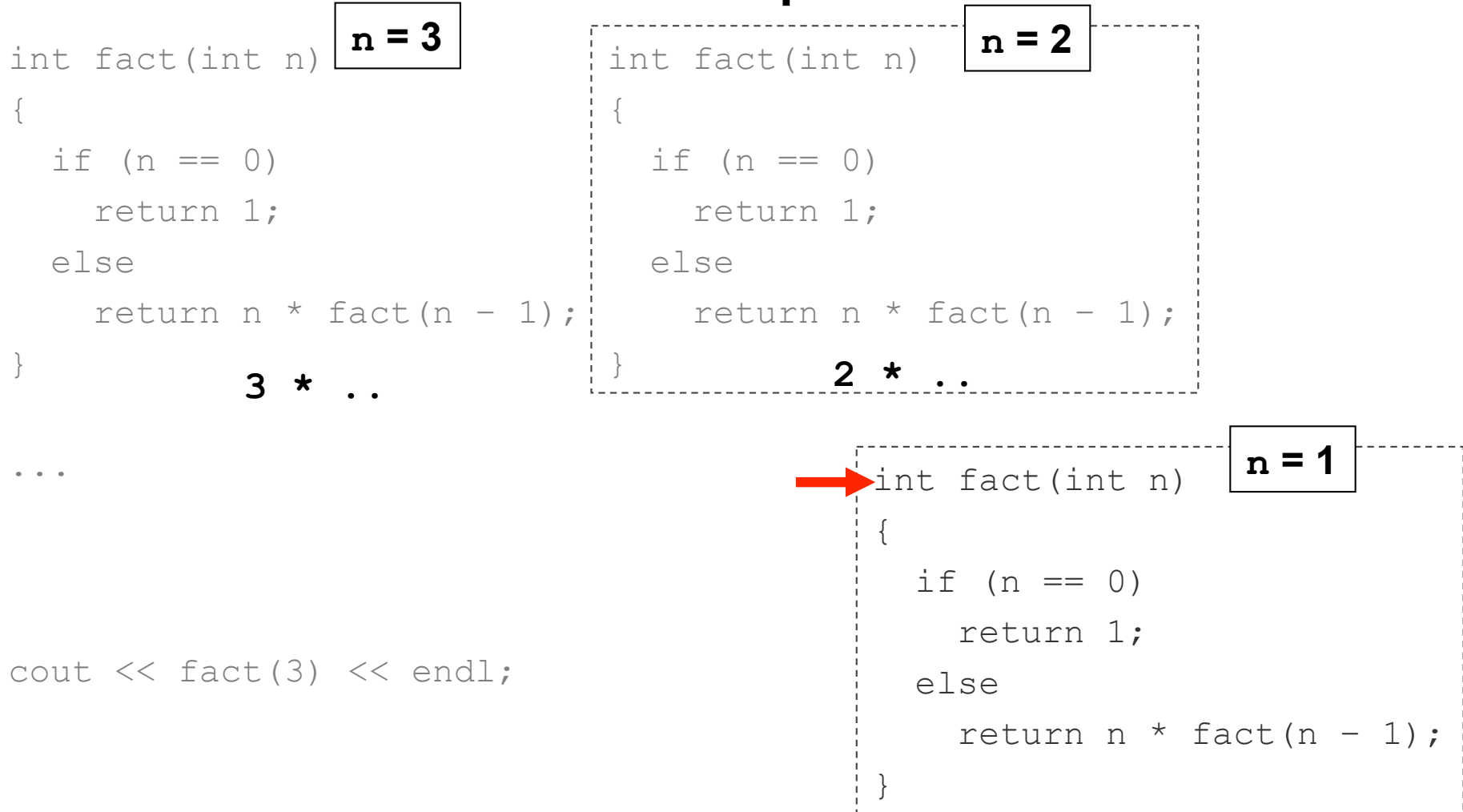
...

```
cout << fact(3) << endl;
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
```

Ici aussi, la condition `n == 0` est fausse, et la fonction `fact` est de nouveau appelée.

Pas-à-pas



A l'appel de `fact(n-1)`, tout se passe comme si une nouvelle fonction `fact` était créée, puis appelée en lui passant 1 en paramètre.


Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

...

```
cout << fact(3) << endl;
```

```
int fact(int n) n = 1
{
    if (n == 0)
        return 1;
    else
         return n * fact(n - 1);
}
```

Encore une fois, la condition `n == 0` est fausse, et la fonction `fact` est de nouveau appelée.

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

```
...
→ int fact(int n) n = 0
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
1 * ..
```

```
int fact(int n) n = 1
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
1 * ..
```

Tout se passe comme si une nouvelle fonction `fact` était créée, puis appelée en lui passant 0 en paramètre.

Pas-à-pas

```
int fact(int n) n = 3
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
2 * ..
```

```
..
int fact(int n) n = 0
{
  → if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
1 * ..
```

```
int fact(int n) n = 1
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
1 * ..
```

Cette fois-ci, la condition est vraie...

Pas-à-pas

```
int fact(int n) n = 3
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
2 * ..
```

```
...
int fact(int n) n = 0
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
1 * ..
```

```
int fact(int n) n = 1
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
1 * ..
```

...et c'est l'instruction
return 1;
qui est exécutée.

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

```
..
int fact(int n) n = 0
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
1 * ..
```

```
int fact(int n) n = 1
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
1 * ..
```

On sort donc de la fonction...


Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

...

```
cout << fact(3) << endl;
```

```
int fact(int n) n = 1
{
    if (n == 0)
        return 1;
    else
         return n * fact(n - 1);
}
1 * ..
```

...pour revenir où elle était appelée, c'est-à-dire ici.


Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

...

```
cout << fact(3) << endl;
```

```
int fact(int n) n = 1
{
    if (n == 0)
        return 1;
    else
         return n * fact(n - 1);
}
1 * 1
```

fact(n-1) est remplacé par la valeur retournée par la fonction précédente, c'est-à-dire par la valeur 1.

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

...

```
cout << fact(3) << endl;
```

```
int fact(int n) n = 1
{
    if (n == 0)
        return 1;
    else
        return 1 * 1 = 1;
}
```

La fonction renvoie donc la valeur 1 (= 1 * 1),
et on continue...

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
3 * ..
```

...

```
int fact(int n) n = 2
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}
2 * ..
```

...ici.

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
2 * 1
```

...


fact(n-1) est remplacé par la valeur retournée par la fonction précédente, c'est-à-dire 1.

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
3 * ..
```

```
int fact(int n) n = 2
{
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}
2 * 1 = 1
```



...

La fonction renvoie donc 2 (=2 * 1), et on continue...

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
    → return n * fact(n - 1);
}
```

3 * ..

...

...ici.

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        → return n * fact(n - 1);
}
           3 *      2
```

...

`fact(n-1)` est remplacé par la valeur retournée par la fonction précédente, c'est-à-dire 2.

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n) n = 3
{
    if (n == 0)
        return 1;
    else
        → return n * fact(n - 1);
}
```

$3 * 2 = 6$

...


La fonction renvoie donc finalement 6.

```
cout << fact(3) << endl;
```

Pas-à-pas

```
int fact(int n)
{
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}

...
```

 `cout << fact(3) << endl;`
 6

6 est affiché.

Fonctions récursives: attention à ne pas oublier l'arrêt

Cette fonction ne marche pas:

```
int fact(int n)
{
    return n * fact(n - 1);
}
```

..

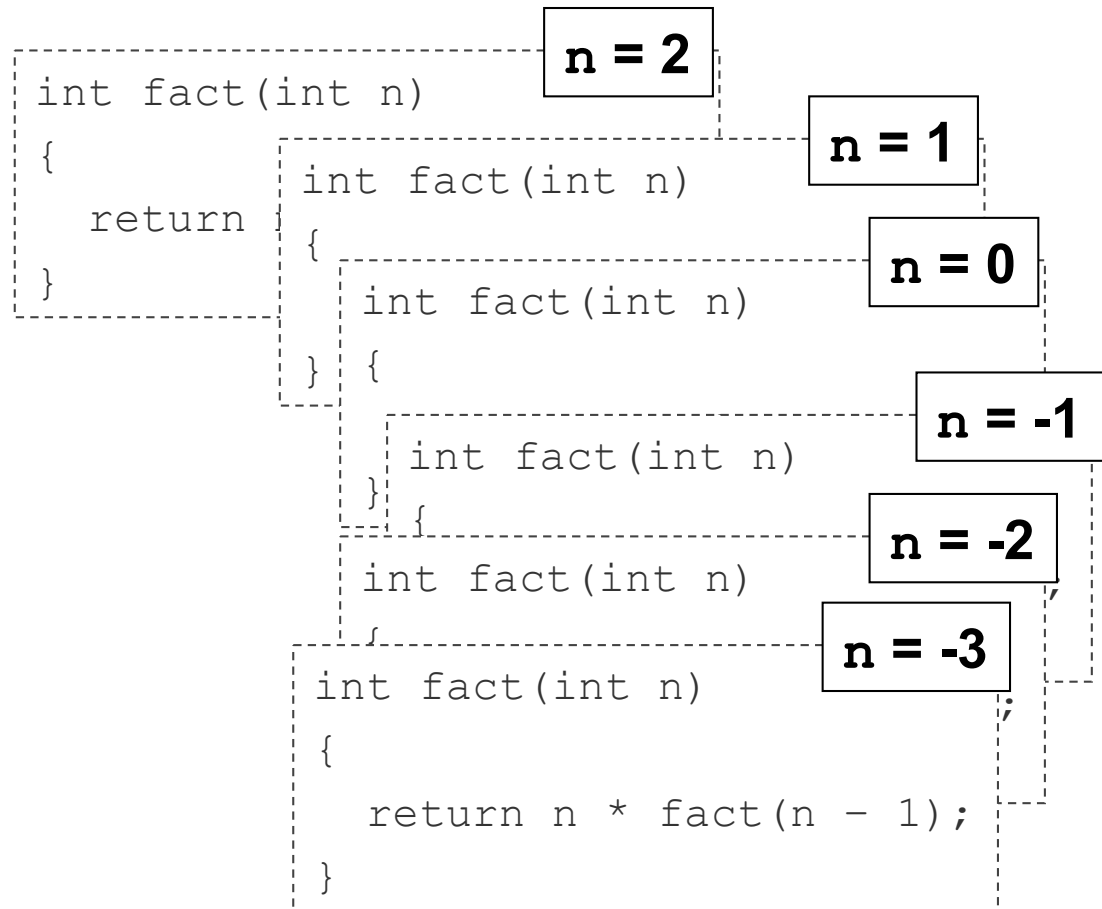
```
cout << fact(3) << endl;
```

Il manque le test d'arrêt. La fonction va continuer de s'appeler.

Attention à ne pas oublier l'arrêt Pas-à-pas

Cette fonction ne marche pas:

```
int fact(int n) n = 3  
{  
    return n * fact(n - 1);  
}  
  
..  
  
cout << fact(3) << endl;
```



etc... Le programme ne s'arrête pas.

A quoi servent les fonctions récursives ?

On aurait aussi pu écrire la fonction `fact` avec une boucle `for`:

```
int fact_avec_for(int n)
{
    int resultat = 1;
    for(int i = 1; i <= n; i++)
        resultat = resultat * i;
    return resultat;
}
```

Cette fonction est appelée une fonction itérative. Les fonctions itératives sont généralement plus rapides que les fonctions récursives.

Pourquoi alors se préoccuper des fonctions récursives ?

Il existe des problèmes qui sont très difficiles à résoudre avec des fonctions itératives, et qui deviennent beaucoup plus faciles quand on utilise des fonctions récursives.

C'est le cas de la fonction `dessine_arbre` que nous allons voir dans la deuxième partie.

Qu'affichent les programmes suivants ?

A.

```
int fact(int n)
{
    cout << "n = " << n << endl;
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}

...

cout << fact(3) << endl;
```

B.

```
void d(int n)
{
    cout << "n = " << n << endl;
    if (n > 0)
    {
        d(n - 1);
        d(n - 1);
    }
}

...

d(3);
```

Graphisme

Zone de dessin

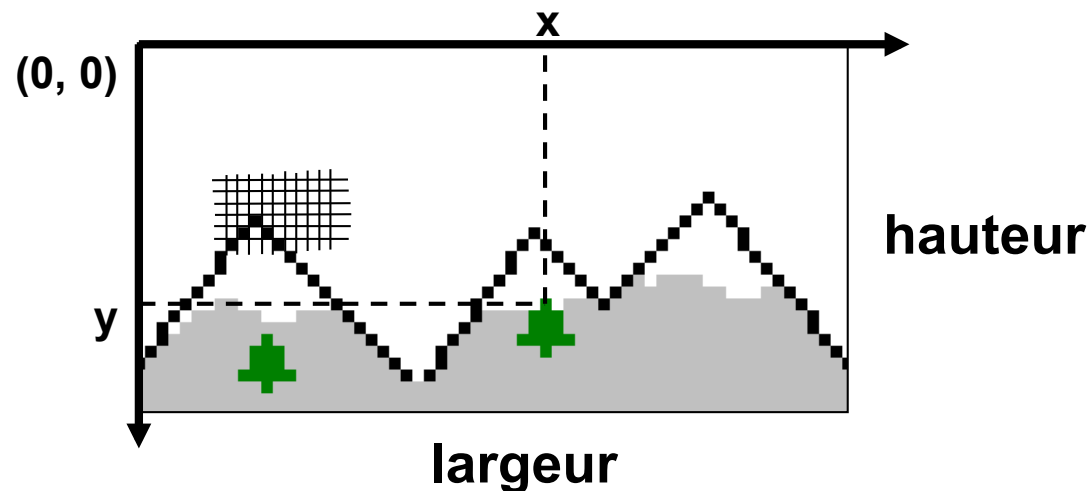
Pour l'ordinateur, une image est une grille rectangulaire dont chaque élément est un *pixel*.

La taille d'une image est définie par:

- sa largeur (le nombre de colonnes), et
- sa hauteur (le nombre de lignes).

Chaque pixel est repéré par

- son abscisse x (le numéro de la colonne), et
- son ordonnée y (le numéro de la ligne).



Attention:

- Contrairement à la convention en mathématiques, l'origine (0,0) est en haut à gauche, et l'axe des ordonnées est vers le bas;
- les coordonnées sont forcément positives ou nulles.

Codage de la couleur

Chaque pixel a sa couleur propre.

Les couleurs sont souvent codées suivant le format RVB, qui utilise 3 composantes: Rouge, Vert, Bleu.

Le format RVB est additif. Par exemple:

- la couleur jaune est obtenue avec la composante rouge égale à la composante verte, et une composante bleue nulle.
- le blanc est obtenu en prenant les 3 composantes au maximum de leur valeur.

Dans la bibliothèque logicielle que nous allons utiliser, les composantes ont une valeur qui varie entre 0 et 1:

JAUNE: Composante rouge = 1.; Composante Vert = 1.; Composante Bleu = 0.

BLANC: $R = 1, V = 1, B = 1$.

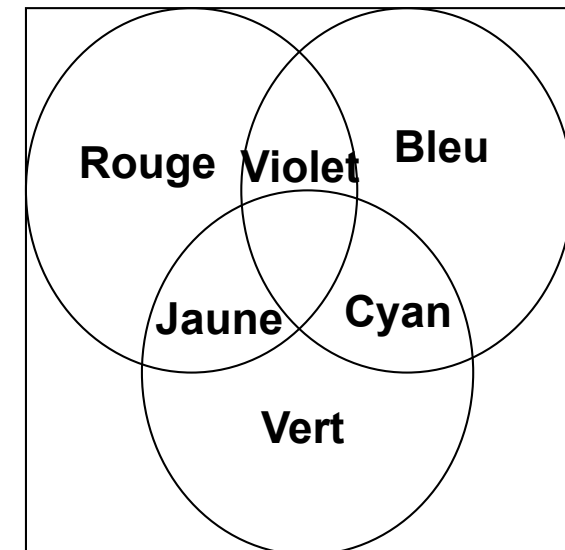
BLEU: $R = 0, 0, 1$.

BLEU PALE: par exemple: $R = 0.5, V = 0.5, B = 1$.

NOIR: $R = 0, V = 0, B = 0$.

GRIS: par exemple: $R = 0.5, V = 0.5, B = 0.5$

etc...



Ouvrir une fenêtre SimpleWindow

Nous allons utiliser une bibliothèque graphique qui dispose de quelques fonctions simples. Cette bibliothèque est écrite en C++, et nous allons prendre un peu d'avance sur le cours pour pouvoir l'utiliser.

Pour pouvoir utiliser ces fonctions, il faut tout d'abord ajouter le `include` suivant:

```
#include "swindow.h"
```

La ligne suivante déclare une variable appelée `window`, de type `SimpleWindow`.

```
SimpleWindow window("dessin", 300, 200);
```

qui va permettre d'ouvrir une fenêtre graphique

- nommée "dessin",
- de largeur 300 colonnes,
- de hauteur 200 lignes.

La fenêtre s'ouvre quand la fonction `map()` est appelée:

```
window.map();
```

Instructions de dessin

1. Il faut d'abord spécifier la couleur courante:

→ Appeler la fonction `color()` en lui donnant en paramètres les composantes rouge, vert et bleu de la couleur utilisée par la suite.

Par exemple, pour définir la couleur bleue:

```
window.color(0, 0, 1);
```

2. Dessiner:

→ Par exemple, pour modifier un pixel, appeler la fonction `drawPoint()` en lui donnant en paramètre les coordonnées x, y du pixel. Pour colorier le pixel aux coordonnées 100, 50:

```
window.drawPoint(100, 50);
```

3. Mettre à jour la fenêtre:

Tant que la fonction `show()` n'est pas appelée, les modifications ne sont pas visibles. L'instruction suivante met la fenêtre à jour:

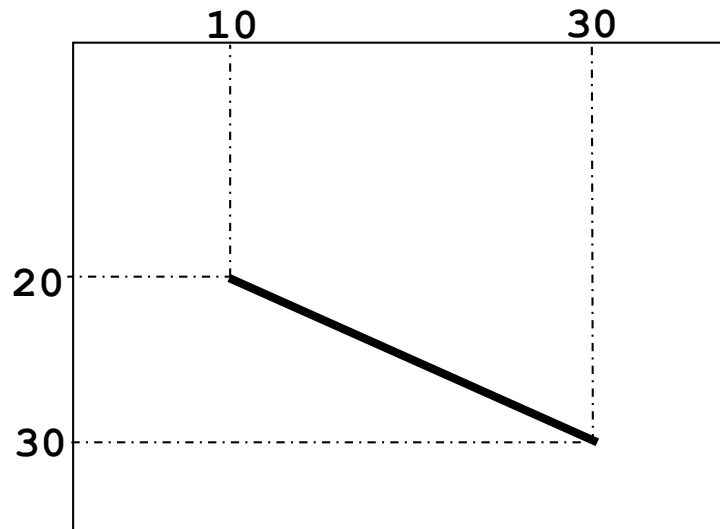
```
window.show();
```

Tracer un segment de droite

Utiliser la fonction `drawLine()` en lui passant en paramètres les coordonnées des deux extrémités.

Exemple: Pour dessiner un segment de droite entre les pixels (10, 20) et (30, 30):

```
window.drawLine(10, 20, 30, 30);
```

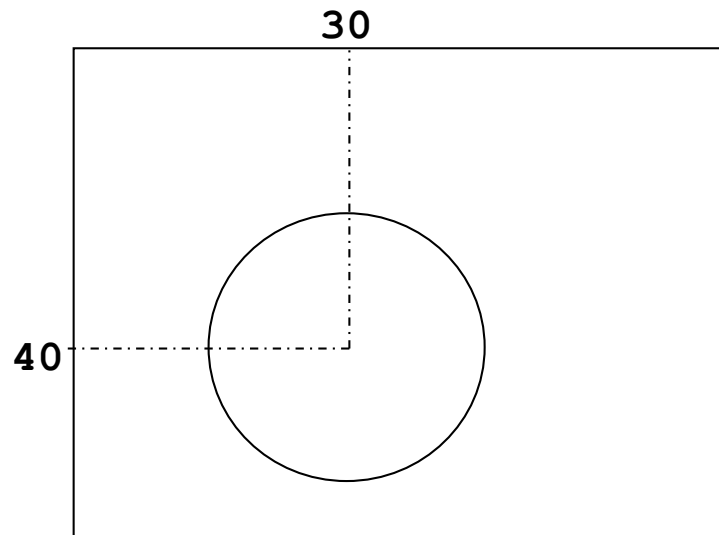


Tracer un cercle

Appeler la fonction `drawCircle()` en lui passant en paramètres le centre du cercle et le rayon.

Exemple: pour dessiner un cercle de centre (30, 40), de rayon 15 pixels:

```
window.drawCircle(30, 40, 15);
```



Autres instructions

Chaînes de caractères:

L'instruction suivante affiche la chaîne "hello" à partir du pixel 10, 50:

```
window.drawText("hello", 10, 50);
```

Rectangle:

L'instruction suivante remplit le rectangle de coin supérieur gauche (20, 30), de largeur 10 pixels, et de longueur 50 pixels:

```
window.fillRect(20, 30, 10, 50);
```

Effacer/Remplir la fenêtre:

L'instruction suivante remplit toute la fenêtre de la couleur courante:

```
window.fill();
```

Mettre à jour la fenêtre

Ne pas oublier d'appeler la fonction `show()` pour mettre à jour la fenêtre !

```
window.show();
```

Tant que `window.show();` n'est pas appelée, les modifications (`drawLine`, `drawCircle...`) ne sont pas visibles.

```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

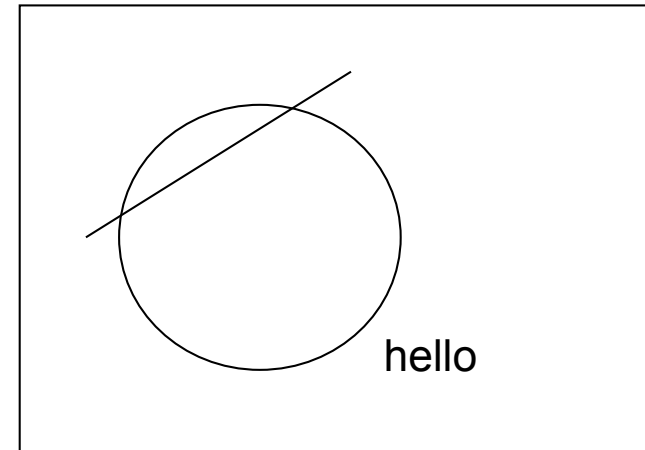
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
getchar();

    return 0;
}
```

```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```

```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
→ SimpleWindow window("dessin", 300, 200);
  window.map();

  // Remplit la fenetre en blanc:
  window.color(1, 1, 1);
  window.fill();

  // Dessine en noir:
  window.color(0, 0, 0);

  // Instructions de dessin:
  window.drawLine(50, 100, 200, 30);
  window.drawCircle(130, 100, 50);
  window.drawText("hello", 170, 150);

  window.show();

  // Attend que l'utilisateur appuie sur une touche:
  getchar();

  return 0;
}
```

```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    → window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    → window.color(1, 1, 1);
    window.fill();

    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    → window.fill();

    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

    // Dessine en noir:
    → window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

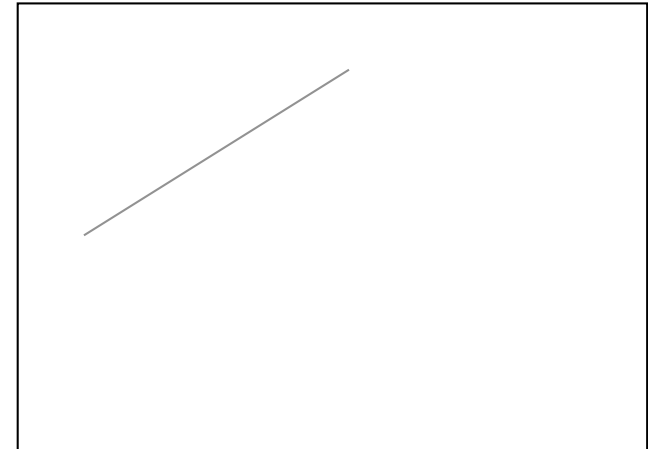
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    → window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

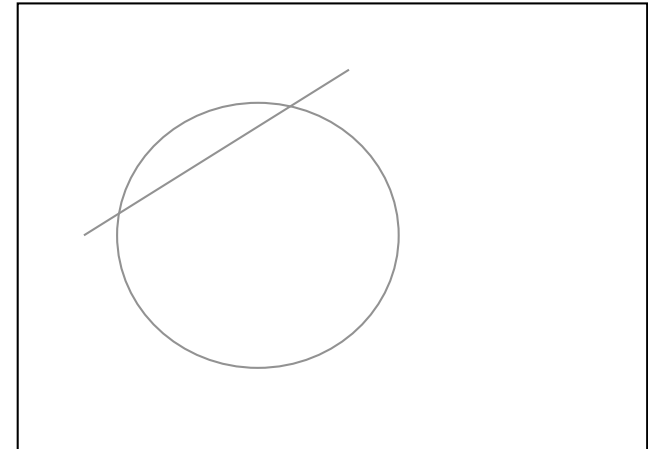
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    → window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

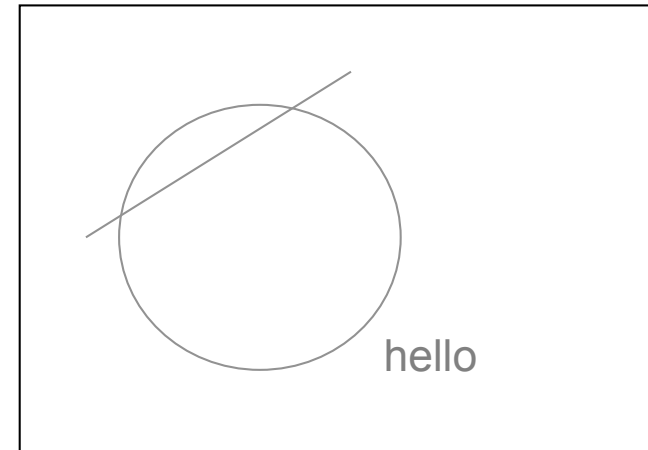
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    → window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

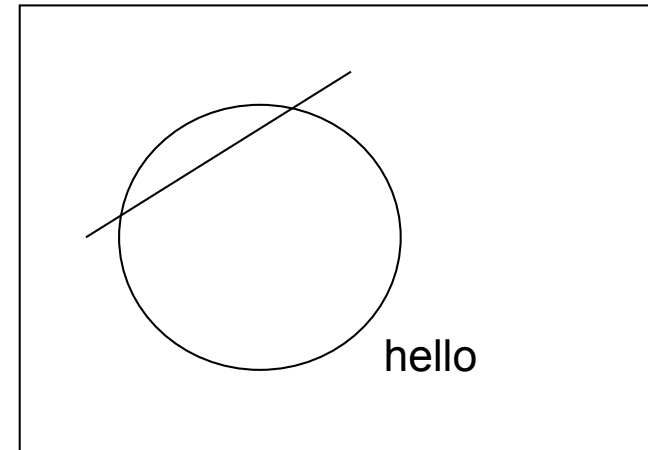
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    → window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

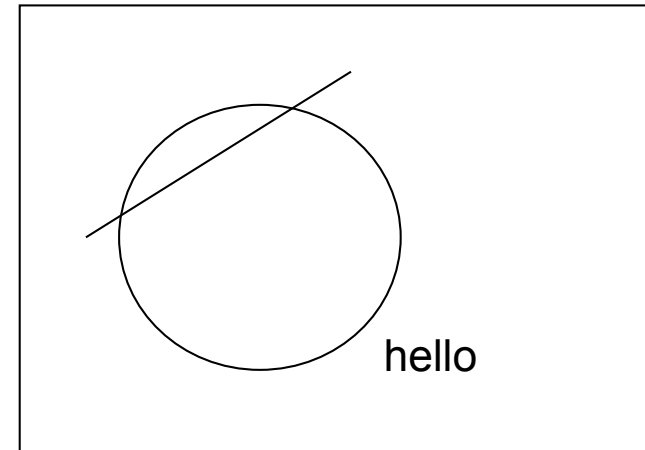
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    → getchar();

    return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

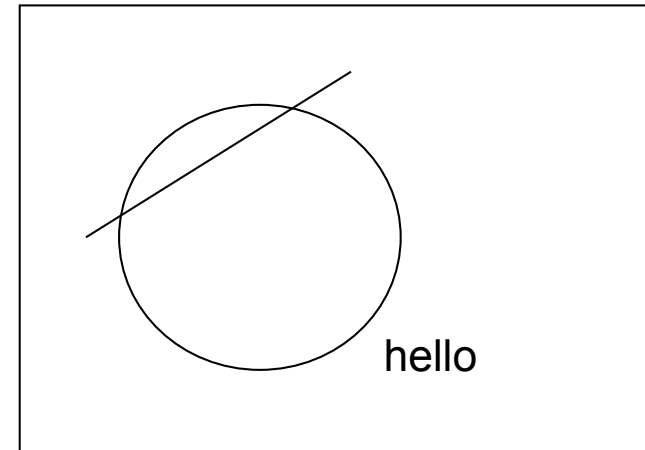
    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    → return 0;
}
```



```
#include <stdio.h> // pour utiliser getchar()
#include "swindow.h"

int main(int argc, char ** argv)
{
    SimpleWindow window("dessin", 300, 200);
    window.map();

    // Remplit la fenetre en blanc:
    window.color(1, 1, 1);
    window.fill();

    // Dessine en noir:
    window.color(0, 0, 0);

    // Instructions de dessin:
    window.drawLine(50, 100, 200, 30);
    window.drawCircle(130, 100, 50);
    window.drawText("hello", 170, 150);

    window.show();

    // Attend que l'utilisateur appuie sur une touche:
    getchar();

    return 0;
}
```

Attention: La ligne

`getchar () ;`

du programme précédente est importante en pratique.

L'appel à la fonction `getchar()` bloque le programme tant que l'utilisateur n'appuie pas sur une touche.

Quand le programme se termine, la fenêtre est fermée automatiquement.

Donc, si vous oubliez le `getchar()`, le programme ouvre la fenêtre, et la referme juste après, en général si rapidement qu'on n'a pas le temps de la voir...

Un autre exemple simple

```
const int longueur = 300;  
const int hauteur = 200;  
const int rayon_max = 50;
```

```
SimpleWindow window("cercles", longueur, hauteur);  
window.map();
```

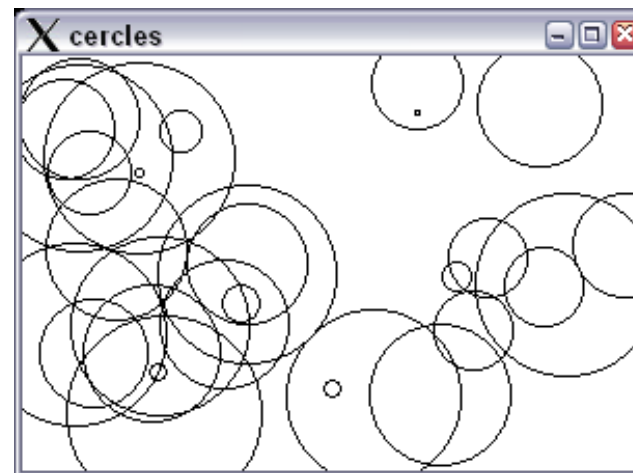
```
// Remplit la fenetre en blanc:  
window.color(1, 1, 1);  
window.fill();
```

```
// Dessine en noir:  
window.color(0, 0, 0);
```

```
for(int i = 0; i < 30; i++)  
    window.drawCircle(rand() % longueur, rand() % hauteur,  
                      rand() % rayon_max);
```

```
window.show();
```

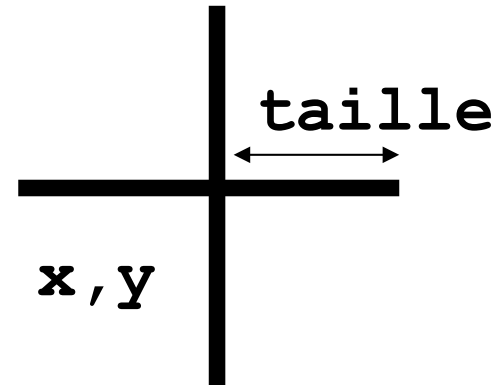
```
getchar();
```



Passer la fenêtre en paramètre

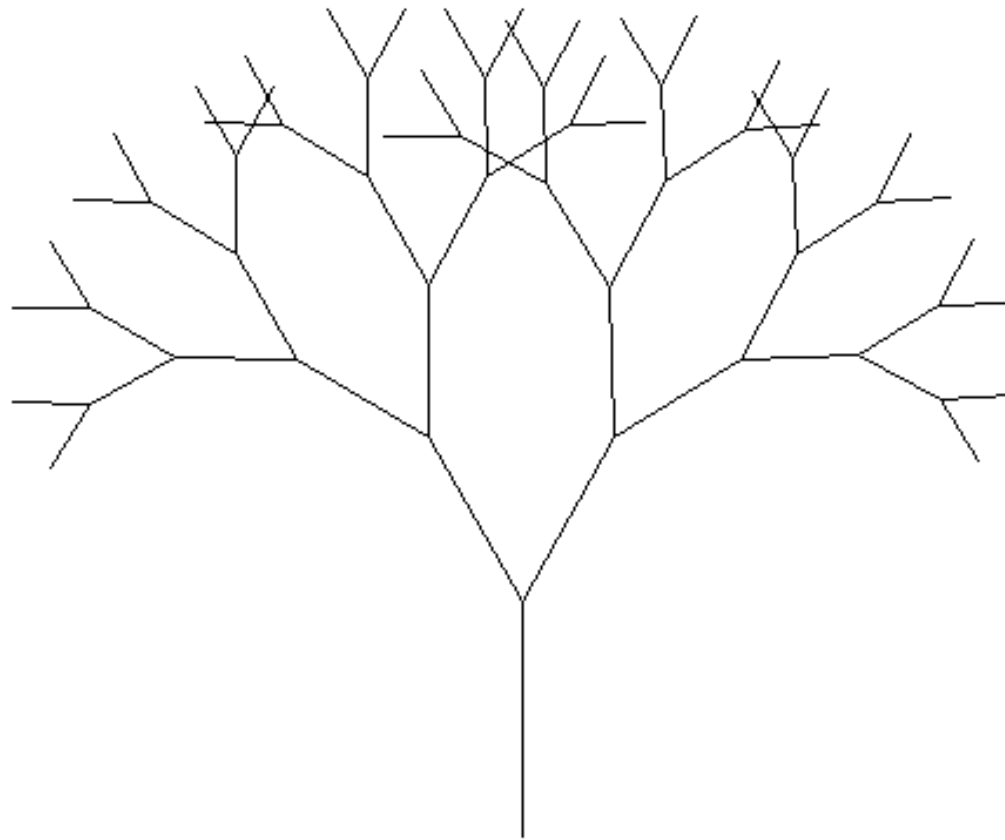
```
void dessine_croix(SimpleWindow * window, int x, int y, int taille)  
{  
    window->drawLine(x - taille, y, x + taille, y);  
    window->drawLine(x, y - taille, x, y + taille);  
}
```

```
int main(...)  
{  
    SimpleWindow window(...);  
  
    ...  
  
    dessine_croix(&window, 100, 50, 5);
```

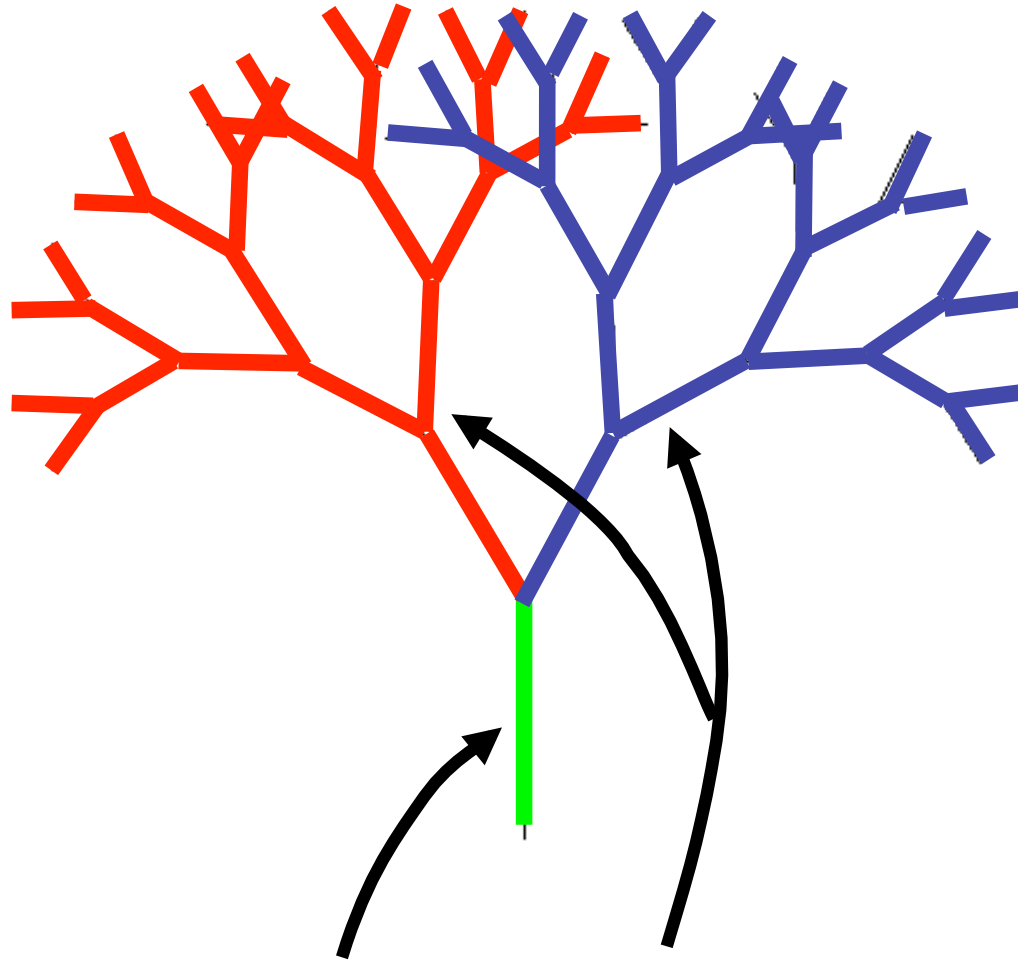


Dessiner un arbre

Comment dessiner un arbre similaire à celui-ci ?



Dessiner un arbre avec une fonction réursive



On peut voir l'arbre comme: un tronc + deux arbres plus petits

Fonction récursive

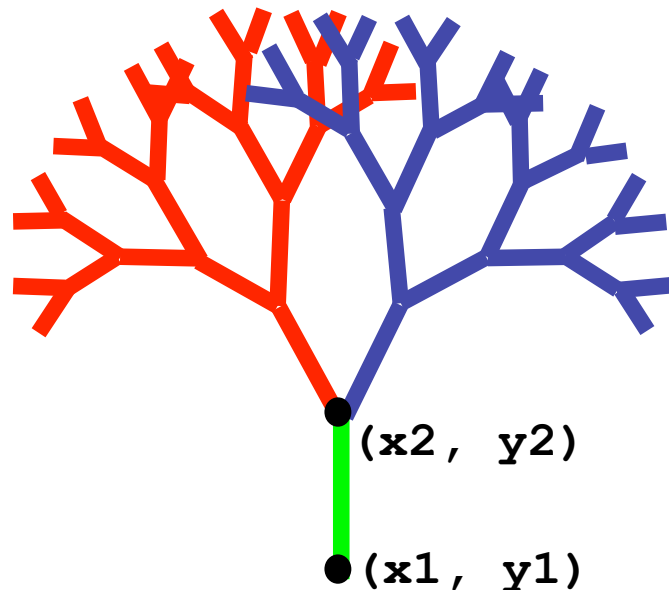
Nous allons écrire une fonction récursive pour dessiner l'arbre.

Cette fonction aura l'en-tête suivant:

```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)
```

où

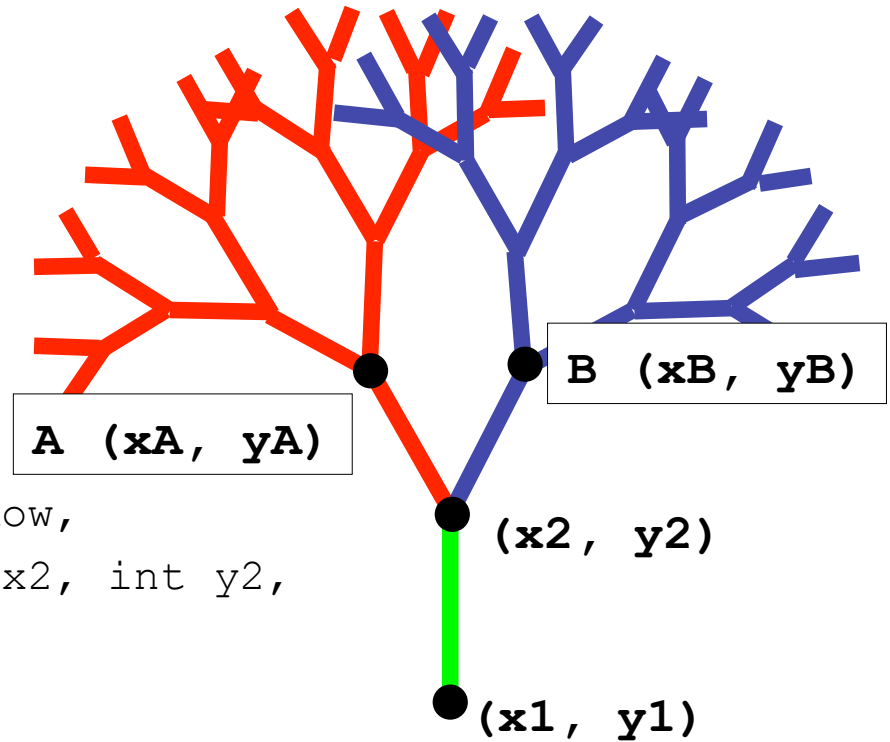
- $(x1, y1)$ et $(x2, y2)$ définissent la position du tronc;
- `nb_niveaux` définit le nombre de fois que la fonction devra s'appeler.



`nb_niveaux = 5` pour l'arbre complet;

`nb_niveaux = 4` pour chacun des "petits" arbres.

Fonction récursive



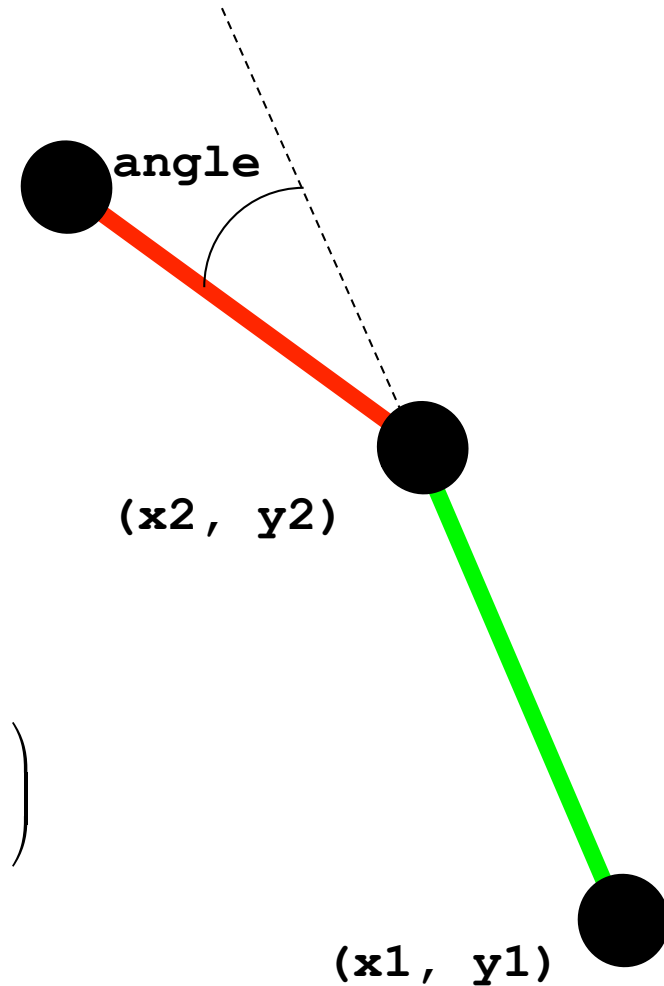
```
void dessine_arbre(SimpleWindow * window,
                  int x1, int y1, int x2, int y2,
                  int nb_niveaux)
{
    // Dessine le tronc:
    window->drawLine(x1, y1, x2, y2);

    if (nb_niveaux > 0) {
        // Calculer  $x_A, y_A$  et  $x_B, y_B$  ...

        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);
    }
}
```

Comment calculer x_A, y_A et x_B, y_B ?

A (x_A, y_A)



$$\begin{pmatrix} x_A \\ y_A \end{pmatrix} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + k \text{Rotation}(\text{angle}) \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

$$\begin{pmatrix} x_A \\ y_A \end{pmatrix} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} + k \begin{pmatrix} \cos(\text{angle}) & -\sin(\text{angle}) \\ \sin(\text{angle}) & \cos(\text{angle}) \end{pmatrix} \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

Fonction récursive

```
void dessine_arbre(SimpleWindow * window,
                  int x1, int y1, int x2, int y2,
                  int nb_niveaux)
{
    // Dessine le tronc:
    window->drawLine(x1, y1, x2, y2);

    if (nb_niveaux > 0)
    {
        int vx = x2 - x1, vy = y2 - y1;
        float angle = 30 * 3.14159 / 180, k = 0.8;

        int xA = int( x2 + k * (cos(angle) * vx - sin(angle) * vy) );
        int yA = int( y2 + k * (sin(angle) * vx + cos(angle) * vy) );
        int xB = int( x2 + k * (cos(-angle) * vx - sin(-angle) * vy) );
        int yB = int( y2 + k * (sin(-angle) * vx + cos(-angle) * vy) );

        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);
    }
}
```

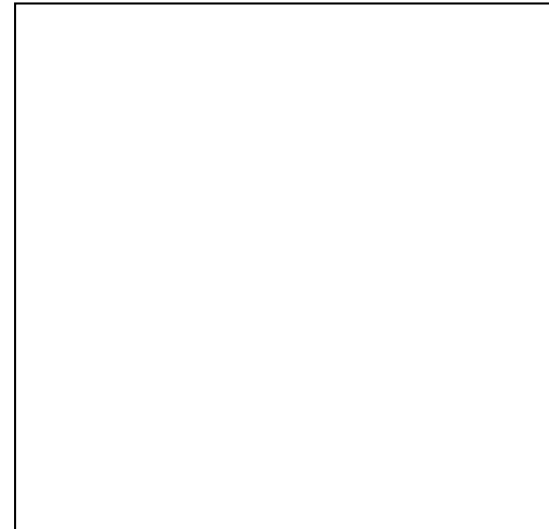
Pas-à-pas

```
void dessine_arbre(SimpleWindow * window,
                  int x1, int y1, int x2, int y2,
                  int nb_niveaux)
{
    // Dessine le tronc:
    window->drawLine(x1, y1, x2, y2);

    if (nb_niveaux > 0)
    {
        [...]

        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);
    }
}
```

```
→ dessine_arbre(&window,
                100, 200, 100, 100,
                2);
```



nb_niveaux = 2



```
void dessine_arbre(SimpleWindow * window,
                  int x1, int y1, int x2, int y2,
                  int nb_niveaux)
{
    // Dessine le tronc:
    window->drawLine(x1, y1, x2, y2);

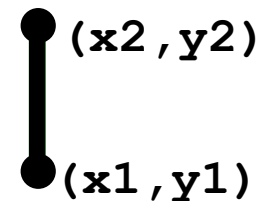
    if (nb_niveaux > 0)
    {
        [...]

        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);
    }
}
```

```
dessine_arbre(&window,
              100, 200, 100, 100,
              2);
```

nb_niveaux = 2

```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)  
{  
    // Dessine le tronc:  
    window->drawLine(x1, y1, x2, y2);  
  
    if (nb_niveaux > 0)  
    {  
        [...]  
  
        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);  
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);  
    }  
}
```



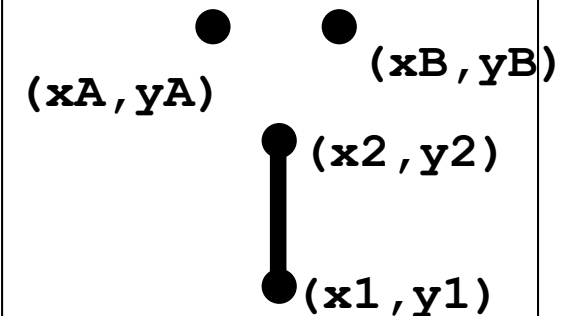
```
dessine_arbre(&window,  
              100, 200, 100, 100,  
              2);
```

nb_niveaux = 2

```
void dessine_arbre(SimpleWindow * window,
                  int x1, int y1, int x2, int y2,
                  int nb_niveaux)
{
    // Dessine le tronc:
    window->drawLine(x1, y1, x2, y2);

    if (nb_niveaux > 0)
    {
        [...]

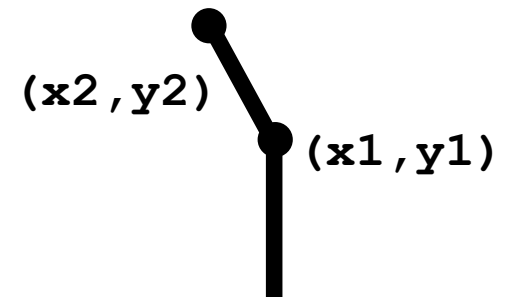
        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);
    }
}
```



```
dessine_arbre(&window,
              100, 200, 100, 100,
              2);
```

nb_niveaux = 1

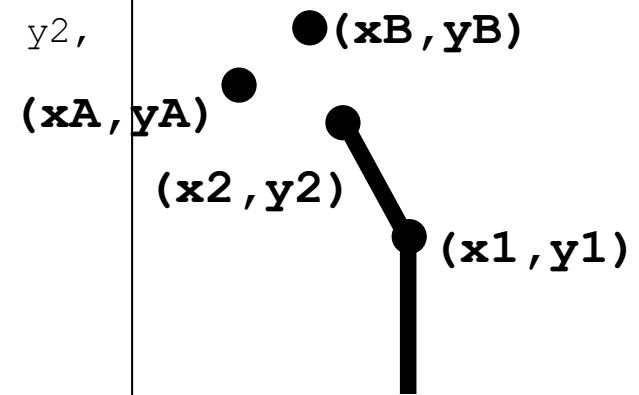
```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)  
{  
    // Dessine le tronc:  
    window->drawLine(x1, y1, x2, y2);  
  
    if (nb_niveaux > 0)  
    {  
        [...]  
  
        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);  
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);  
    }  
}
```



```
dessine_arbre(&window,  
             100, 200, 100, 100,  
             2);
```

nb_niveaux = 1

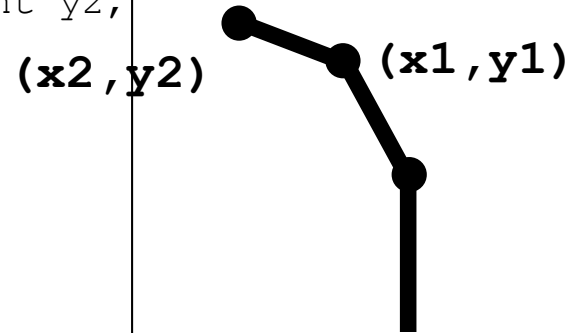
```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)  
{  
    // Dessine le tronc:  
    window->drawLine(x1, y1, x2, y2);  
  
    if (nb_niveaux > 0)  
    {  
        [...]  
  
        → dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);  
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);  
    }  
}
```



```
dessine_arbre(&window,  
              100, 200, 100, 100,  
              2);
```

nb_niveaux = 0

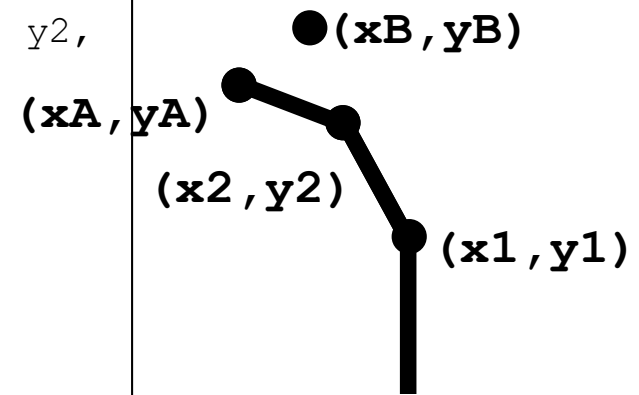
```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)  
{  
    // Dessine le tronc:  
    window->drawLine(x1, y1, x2, y2);  
  
    if (nb_niveaux > 0)  
    {  
        [...]  
  
        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);  
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);  
    }  
}
```



```
dessine_arbre(&window,  
             100, 200, 100, 100,  
             2);
```

nb_niveaux = 1

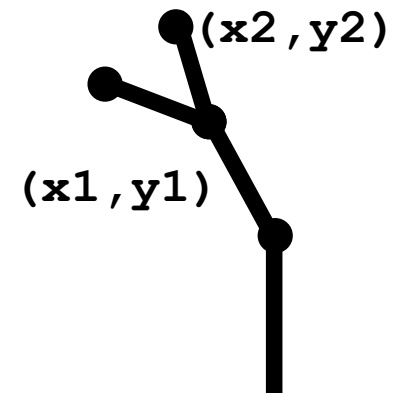
```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)  
{  
    // Dessine le tronc:  
    window->drawLine(x1, y1, x2, y2);  
  
    if (nb_niveaux > 0)  
    {  
        [...]  
  
        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);  
        → dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);  
    }  
}
```



```
dessine_arbre(&window,  
              100, 200, 100, 100,  
              2);
```

nb_niveaux = 0

```
void dessine_arbre(SimpleWindow * window,  
                  int x1, int y1, int x2, int y2,  
                  int nb_niveaux)  
{  
    // Dessine le tronc:  
    window->drawLine(x1, y1, x2, y2);  
  
    if (nb_niveaux > 0)  
    {  
        [...]  
  
        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);  
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);  
    }  
}
```

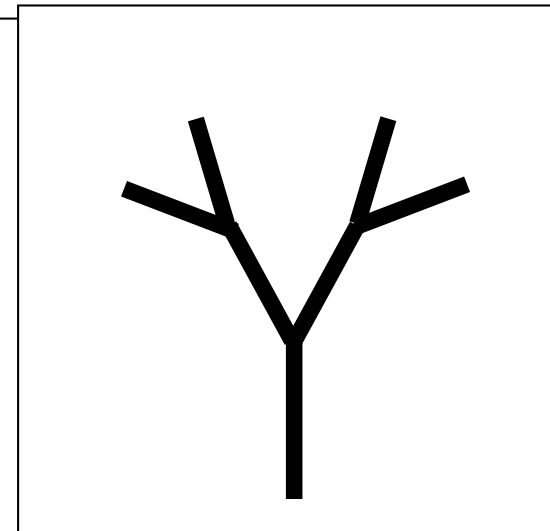


```
dessine_arbre(&window,  
              100, 200, 100, 100,  
              2);
```

```
void dessine_arbre(SimpleWindow * window,
                  int x1, int y1, int x2, int y2,
                  int nb_niveaux)
{
    // Dessine le tronc:
    window->drawLine(x1, y1, x2, y2);

    if (nb_niveaux > 0)
    {
        [...]

        dessine_arbre(window, x2, y2, xA, yA, nb_niveaux - 1);
        dessine_arbre(window, x2, y2, xB, yB, nb_niveaux - 1);
    }
}
```

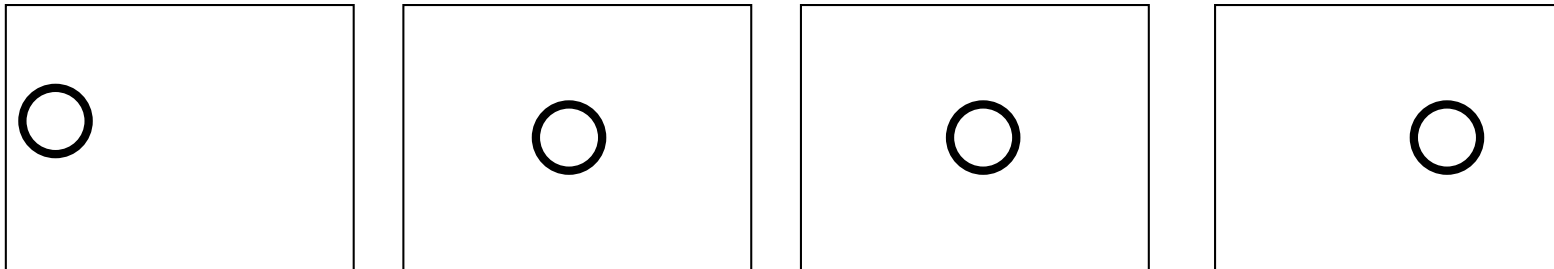


```
dessine_arbre(&window,
             100, 200, 100, 100,
             2);
```

Animations

On peut également réaliser des animations.

Le principe est similaire à celui d'un dessin animé: il suffit d'afficher les images successives suffisamment rapidement pour créer une impression de mouvement:



Animation

Créer la fenêtre:

```
SimpleWindow window("nom", longueur, hauteur);  
window.map();
```

Construire l'image en mémoire:

```
window.color(1, 1, 1);  
window.fill();  
...
```

Afficher l'image à l'écran:

```
window.show();
```

Attendre un peu:

```
usleep(10);
```

Passer à l'image suivante:

La fenêtre est effacée à chaque image.

L'instruction `window.show();` est placée dans la boucle pour mettre à jour la fenêtre à chaque fois que la droite est dessinée.

La fonction `usleep()` permet de stopper temporairement le programme, de façon à voir l'image.

Animation

Le code suivant anime un cercle se déplaçant horizontalement:

```
for(int i = 0; i < largeur; i++)
{
    // Construire l'image:
    window.color(1, 1, 1);
    window.fill();
    // dessine un cercle noir, centre sur la colonne i:
    window.color(0, 0, 0);
    window.drawCircle(i, 200, 50);

    // Afficher l'image à l'écran:
    window.show();

    // Attendre un peu:
    usleep(10);
}
```

Neige

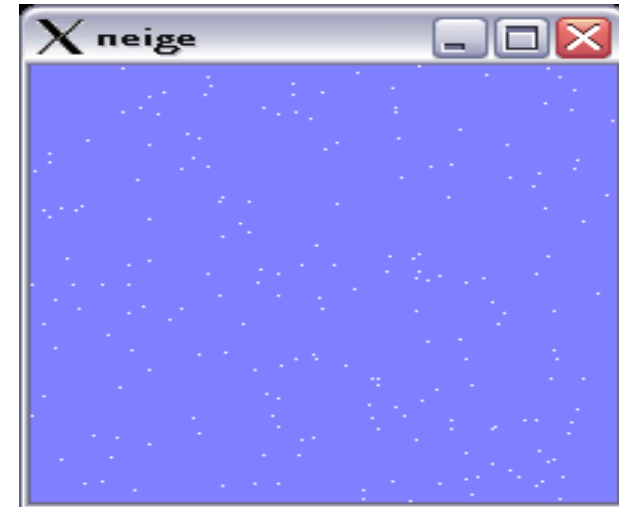
Nous allons maintenant écrire un programme simulant des flocons de neige.

Définissons d'abord définir une structure Flocon:

```
struct Flocon
{
    int x, y;
};
```

Nous allons utiliser un tableau de Flocons:

```
const int nb_flocons = 500;
Flocon flocons[nb_flocons];
```



Vue d'ensemble

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
}
```

Vue d'ensemble (suite)

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;
```

```
struct Flocon
{
    int x, y;
};
```

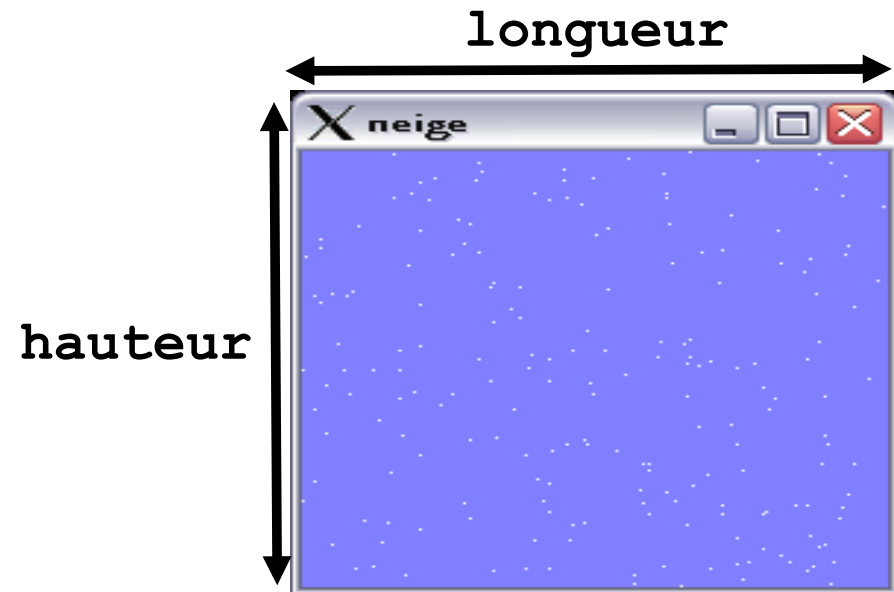
```
// Quelques fonctions utiles
...
```

```
int main(int argc, char ** argv)
{
```

```
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();
```

```
    Flocon flocons[nb_flocons];
```

```
    // Initialiser les flocons:
    ...
```



```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
}
```

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
```

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
```

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
}
```

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
```

```
#include <stdio.h>
#include "swindow.h"

const int longueur = 300;
const int hauteur = 200;
const int nb_flocons = 500;

struct Flocon
{
    int x, y;
};

// Quelques fonctions utiles
...

int main(int argc, char ** argv)
{
    SimpleWindow window("flocons", longueur, hauteur);
    window.map();

    Flocon flocons[nb_flocons];

    // Initialiser les flocons:
    ...
}
```

Vue d'ensemble (suite)

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

```
while(true)
{
    // Construire l'image:
    window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
    window.fill();
    window.color(1, 1, 1); // dessine en blanc
    ... // afficher les flocons

    // Afficher l'image:
    window.show();

    ... // deplacer les flocons

    // Attendre un peu:
    usleep(10);
};

return 0;
}
```

Initialiser les flocons

```
struct Flocon  
{  
    int x, y;  
};
```

...

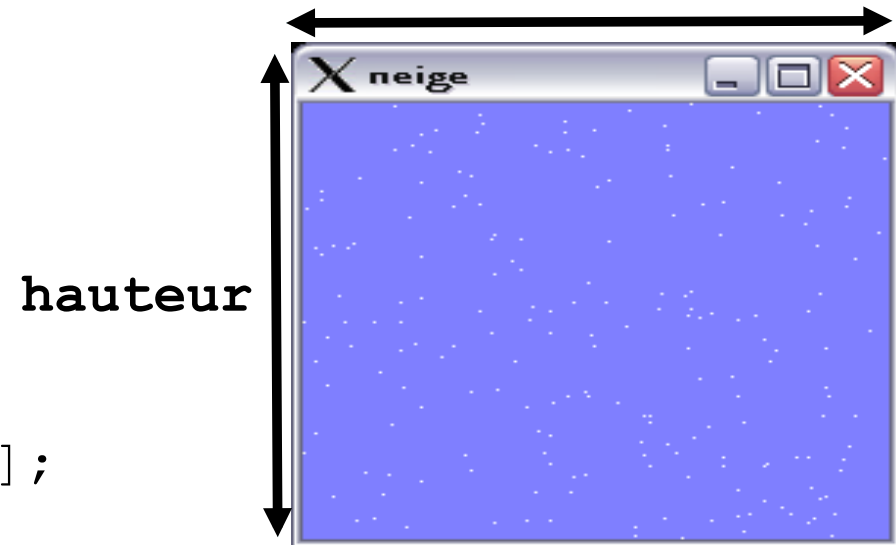
```
Flocon flocons[nb_flocons];
```

```
// Initialiser les flocons:
```

```
for(int i = 0; i < nb_flocons; i++)
```

```
    init_flocon(flocons + i, longueur, hauteur);
```

Pointeur sur le flocon numéro i



init_flocon

```
struct Flocon  
{  
    int x, y;  
};
```

```
// Quelques fonctions utiles
```

```
void init_flocon(Flocon * f, int L, int H)
```

```
{  
    f->x = rand() % L;  
    f->y = rand() % H;  
}
```

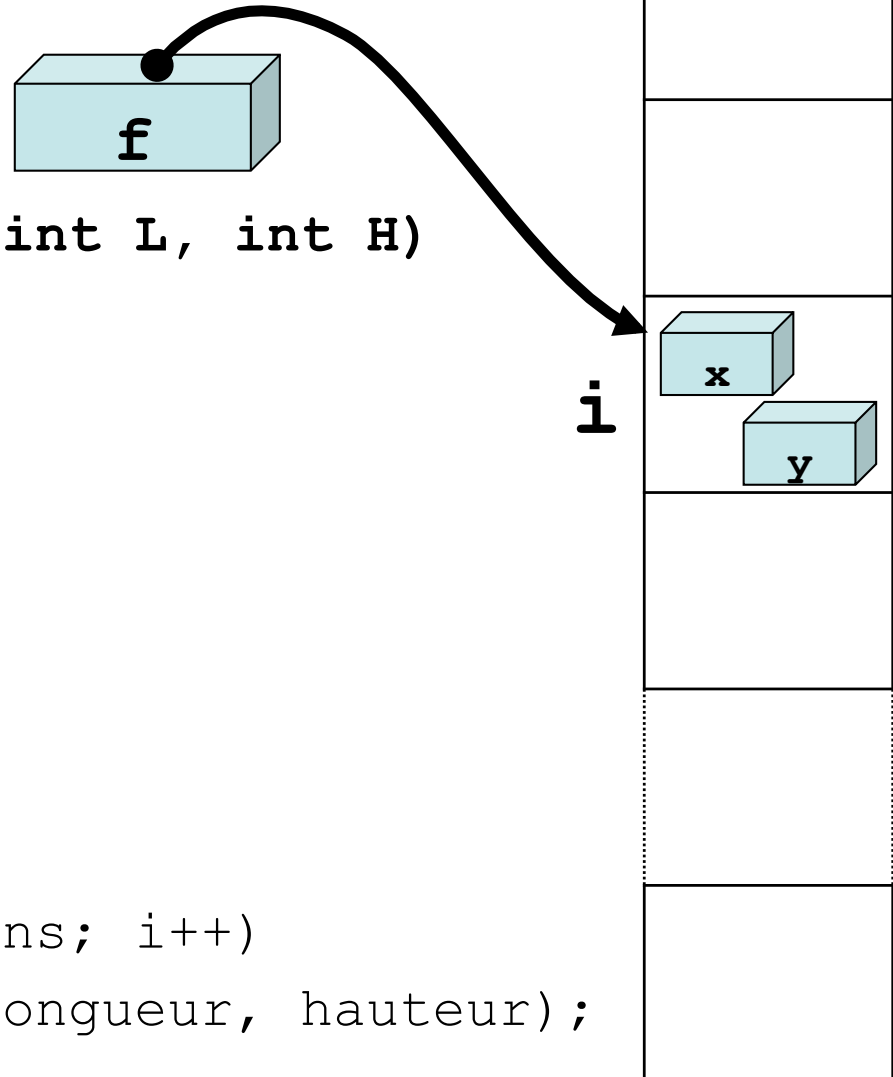
```
...
```

```
Flocon flocons[nb_flocons];
```

```
// Initialiser les flocons:
```

```
for(int i = 0; i < nb_flocons; i++)
```

```
    init_flocon(flocons + i, longueur, hauteur);
```



Afficher les flocons

```
void dessine_flocon(SimpleWindow * window, Flocon * f)
{
    window->drawPoint(f->x, f->y);
}

...

// Construire l'image:
window.color(0.5, 0.5, 1); // efface la fenetre en bleu clair
window.fill();
window.color(1, 1, 1); // dessine en blanc

// Afficher les flocons:
for(int i = 0; i < nb_flocons; i++)
    dessine_flocon(&window, flocons + i);
```

avance_flocon (version 1)

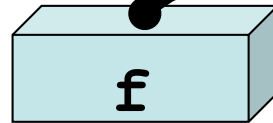
Commençons par modifier uniquement la coordonnée y des flocons à chaque image:

```
void avance_flocon(Flocon * f)
{
    f->y = f->y + 1;
}

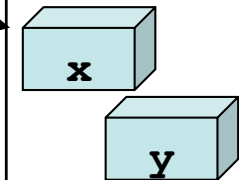
...

// Afficher l'image:
window.show();

// Déplacer les flocons:
for(int i = 0; i < nb_flocons; i++)
    avance_flocon(flocons + i);
```



i



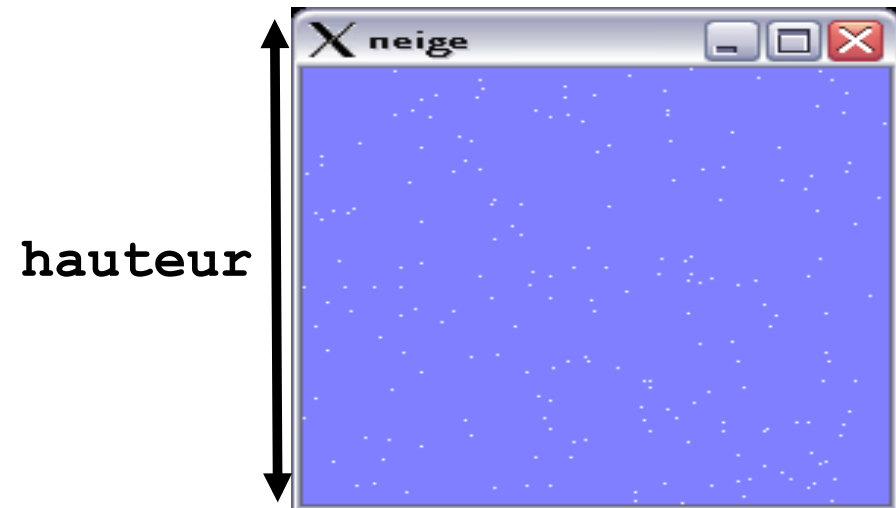
Les flocons tombent verticalement...
...et finissent par sortir de la fenêtre.

avance_flocon (Version 2)

Pour éviter que la fenêtre soit vide après un certain temps, remplaçons un flocon tombé en bas de la fenêtre en haut de la fenêtre:

```
void avance_flocon(Flocon * f, int hauteur)
{
    f->y = f->y + 1;

    if (f->y >= hauteur)
        f->y = 0; // il recommence en haut.
}
```



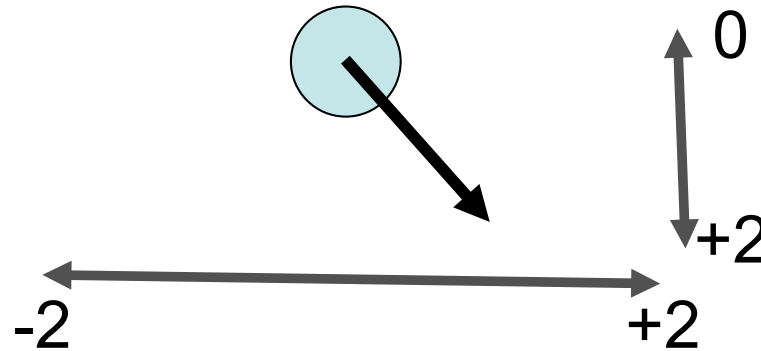
avance_flocon (Version 3)

On peut rendre le déplacement des flocons plus "réaliste":

- en variant la vitesse de chute: au lieu d'ajouter 1 à la coordonnée y à chaque image, lui ajouter une valeur aléatoire (disons entre 0 et 2);
- en modifiant la coordonnée en x , parfois vers la gauche, parfois vers la droite..., en lui ajoutant une valeur aléatoire (disons entre -2 et 2).

```
void avance_flocon(Flocon * f, int hauteur)
{
    f->x = f->x + rand() % 5 - 2; // entre -2 et 2
    f->y = f->y + rand() % 3; // entre 0 et 2

    if (f->y >= hauteur)
        f->y = 0; // il recommence en haut.
}
```



Mise en œuvre

Télécharger le fichier `swindow.tgz` :

lien [swindow.tgz](http://cvlab.epfl.ch/~lepetit/sv) sur la page <http://cvlab.epfl.ch/~lepetit/sv> **dans votre répertoire *programmationI***.

Dans un Terminal:

```
cd programmationI
```

Décompresser le fichier `swindow.tgz` qui crée 3 fichiers (`swindow.cc`, `swindow.h` et `example.cc`):

```
tar zxvf swindow.tgz
```

Compiler `swindow.cc` pour créer le fichier `swindow.o`:

```
g++ -c swindow.cc
```

Compiler l'exemple:

```
g++ swindow.o -L/usr/X11R6/lib -lX11 -o example example.cc
```

Lancer l'exemple:

```
./example
```

Mise en œuvre

```
g++ -c swindow.cc
```

crée un fichier `swindow.o` qui contient le code compilé de la bibliothèque graphique.

Pour créer un exécutable :

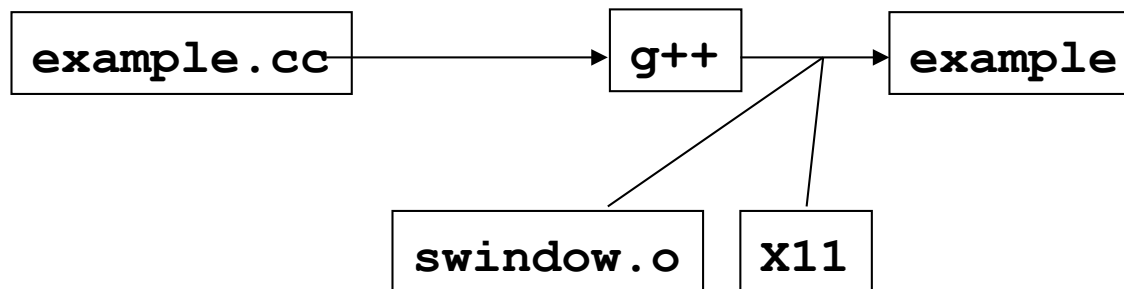
```
g++ [...] -o example example.cc
```

Il faut ajouter `swindow.o` à la compilation:

```
g++ swindow.o [...] -o example example.cc
```

Il faut aussi ajouter `-L /usr/X11R6/lib -lX11`:

```
g++ swindow.o -L/usr/X11R6/lib -lX11 -o example example.cc
```



Mise en œuvre

Pour compiler vos programmes: comme pour compiler `example`, il faut aussi ajouter `swindow.o -L /usr/X11R6/lib -lX11`:

```
g++ swindow.o -L/usr/X11R6/lib -lX11 -o mon_prg mon_prg.cpp
```

flock of birds

A partir de quelques règles très simples modélisant le déplacement d'un individu, on obtient un comportement de groupe complexe. Nous allons considérer un modèle simplifié où un individu tend à s'éloigner des zones trop peuplées, et à se rapprocher des zones peu peuplées.

Un individu sera défini par la structure suivante :

```
struct Oiseau
{
    float x, y;
    float dx, dy;
};
```

où x et y désignent les coordonnées dans le plan 2D de l'oiseau, et dx et dy son vecteur vitesse.

Un essaim sera défini par la structure suivante:

```
struct Essaim
{
    Oiseau * oiseaux;
    int nb_oiseaux;
};
```

où `oiseaux` est un pointeur sur un tableau d'Oiseaux, et `nb_oiseaux` le nombre d'éléments de ce tableau.

Ecrivez la fonction

```
Essaim * alloue_essaim(int n)
```

qui permet d'allouer une structure `Essaim` comportant `n` individus.

Ecrivez la fonction

```
void initialise_oiseau(Oiseau * o, int longueur, int hauteur)
```

qui initialise l'individu pointé par `o`.

Les coordonnées x et y seront tirées aléatoirement entre 0 et longueur, et 0 et hauteur respectivement.

Le vecteur vitesse défini par les champs dx et dy devra avoir une norme égale à 2, et une direction aléatoire.

Pour cela, on tirera un angle α aléatoirement entre 0 et 2Pi , et dx sera initialisée à $2\cos\alpha$, dy à $2\sin\alpha$. La valeur de Pi peut être obtenue avec la constante `M_PI`.

Ecrivez la fonction

```
void initialise_essaim(Essaim * e,  
                        int longueur, int hauteur)
```

qui appelle la fonction

```
void initialise_oiseau(Oiseau * o,  
                       int longueur, int hauteur)
```

pour chacun des individus de l'essaim pointé par e.

Ecrivez la fonction

```
void affiche_essaim(SimpleWindow * window, Essaim * e)
```

qui dessine chacun des individus de `e` dans la fenêtre pointée par `window`. Un individu sera simplement représenté par un point grâce à la fonction `drawPoint`.

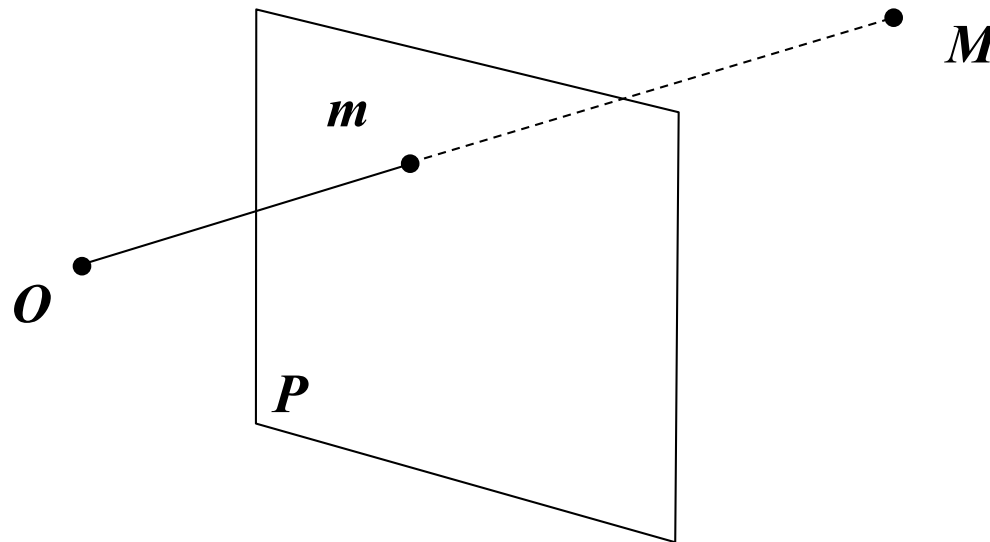
3D

On peut également représenter des objets tri-dimensionnels dans la zone de dessin, il suffit de savoir où doit être affiché dans la fenêtre un point 3D (X, Y, Z) :

Utilisons un modèle de projection perspective:

Soit O le centre de projection, P le plan de projection. L'image m d'un point 3D M est obtenu en trouvant le point d'intersection de la droite (OM) avec P , le plan de projection.

Le plan P va correspondre à la zone de dessin, les coordonnées du point m seront les coordonnées du pixel où apparaît le point M .

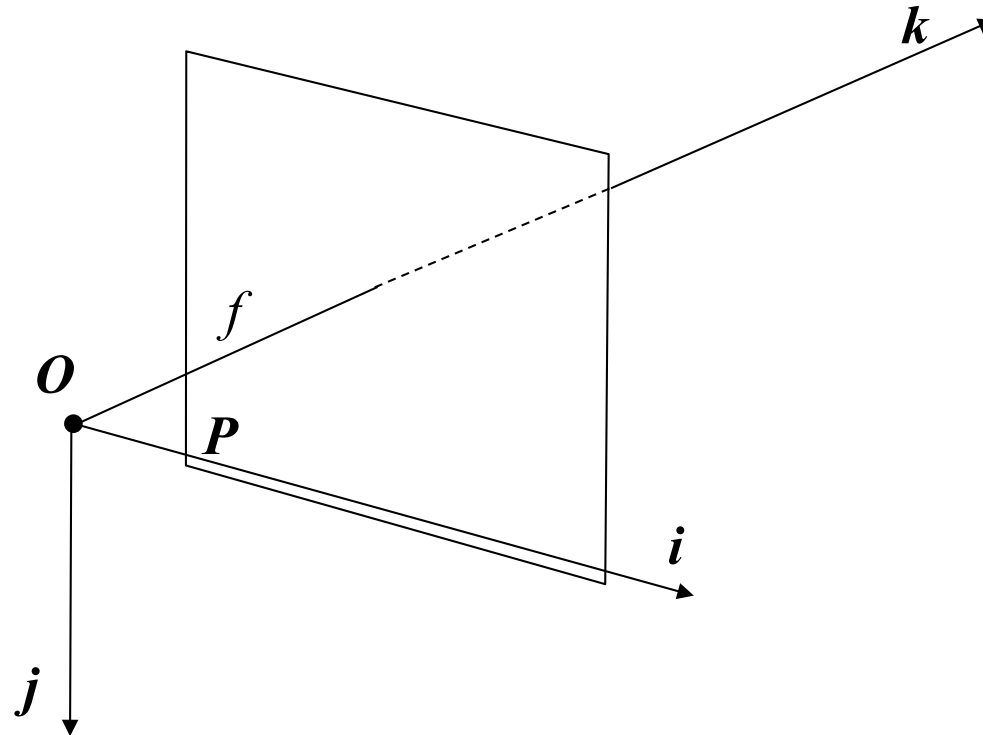


3D

Par commodité, le repère 3D ($Oijk$) peut être choisi tel que:

- le plan (Oij) soit parallèle au plan de projection P ;
- l'axe (Ok) soit perpendiculaire au plan P ;
- tout point de l'axe (Ok) se projette au centre de la zone de dessin;

La distance du point O au plan P est appelée distance focale et est notée f .

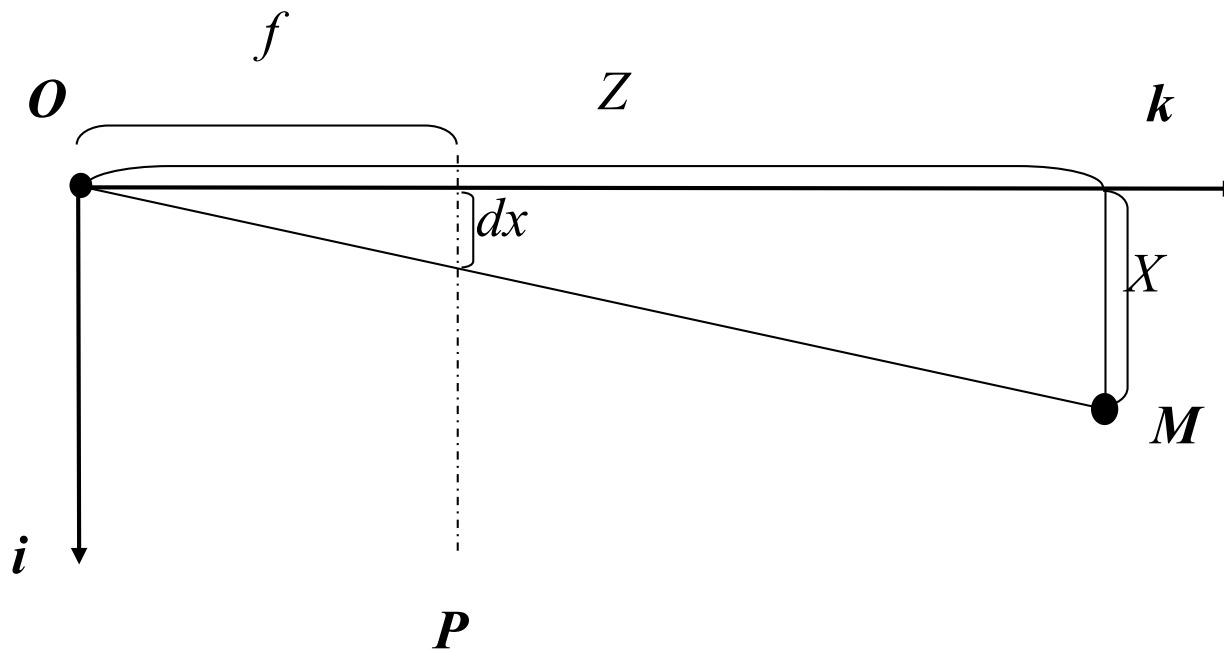


Conversion 3D-2D

La distance dx vérifie $\frac{dx}{f} = \frac{X}{Z}$

$$\text{d'où } dx = f \frac{X}{Z}$$

$$\text{d'où } x = \frac{\text{largeur}}{2} + f \frac{X}{Z}$$



Conversion 3D-2D

La fonction suivante utilise cette formule pour calculer le pixel x , y correspondant au point 3D X , Y , Z :

```
void projection3D2D(float X, float Y, float Z,  
                  float f, int largeur, int hauteur,  
                  float * x, float *y)  
{  
    *x = largeur / 2. + f * X / Z;  
    *y = hauteur / 2. + f * Y / Z;  
}
```

Conversion 3D-2D

Le code suivant dessine une pyramide:

```
const int largeur = 200;
const int hauteur = 200;
const float f = 100;
// ...
projection3D2D(100, 100, 200, f, largeur, hauteur, &x0, &y0);
projection3D2D(-100, 100, 200, f, largeur, hauteur, &x1, &y1);
projection3D2D(-100, 100, 400, f, largeur, hauteur, &x2, &y2);
projection3D2D(100, 100, 400, f, largeur, hauteur, &x3, &y3);

projection3D2D(0, -100, 300, f, largeur, hauteur, &xs, &ys);

window.drawLine(x0, y0, x1, y1);
window.drawLine(x1, y1, x2, y2);
window.drawLine(x2, y2, x3, y3);
window.drawLine(x3, y3, x0, y0);

window.drawLine(x0, y0, xs, ys);
window.drawLine(x1, y1, xs, ys);
window.drawLine(x2, y2, xs, ys);
window.drawLine(x3, y3, xs, ys);
```

