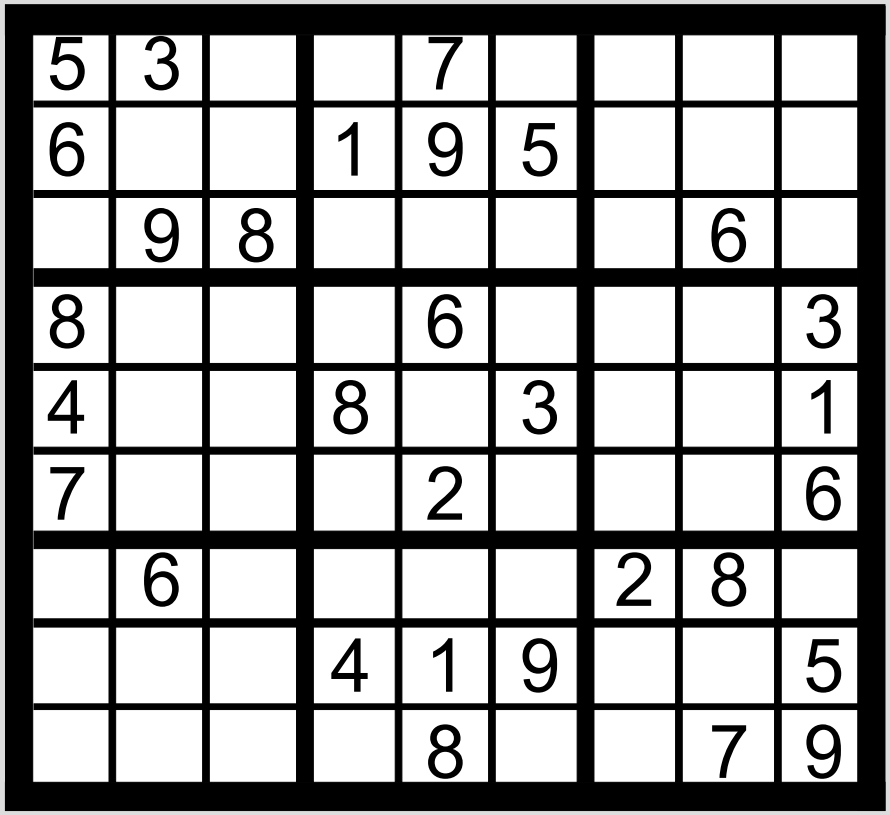
PRÉSENTATION DE *SUDOKUSC*

Comment réaliser un programme capable de compléter n'importe quelle grille de Sudoku ?



- L'algorithme utilisé (la méthode)
- Son implémentation dans un programme (la programmation)
- Le résultat final (le programme)

Choix d'une méthode

Quels critères permettent de choisir la méthode de résolution à utiliser ?

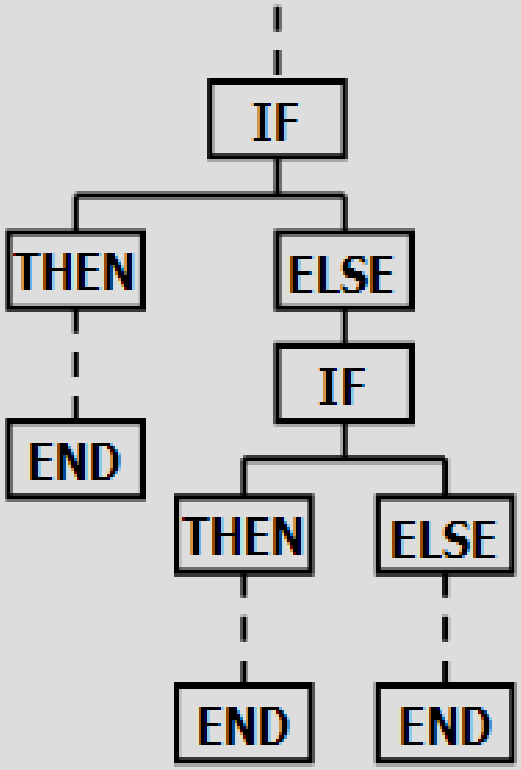
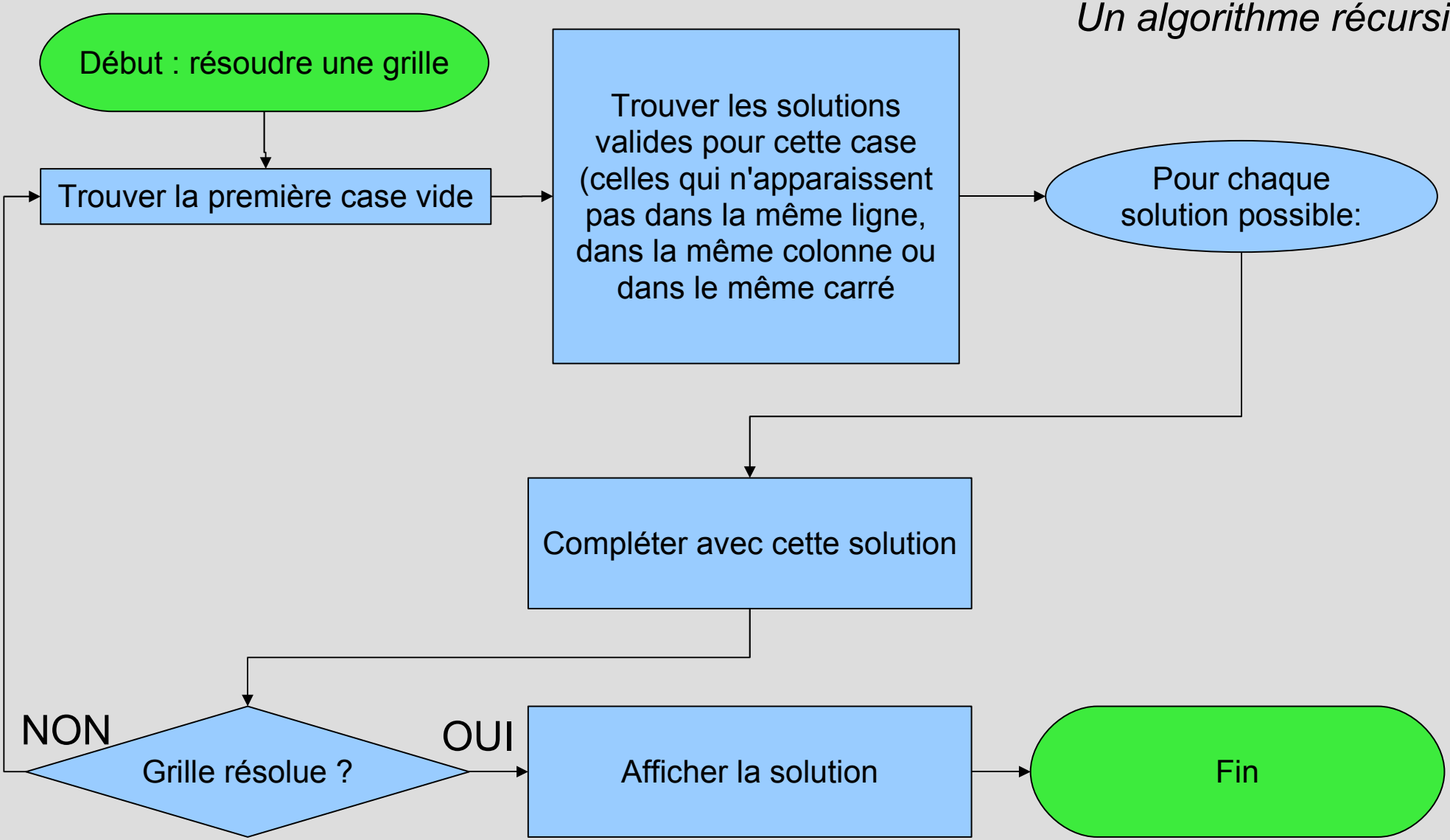


Image de Splintax sur Wikimedia Commons, licence GNU GFDL

- Simplicité (répétition d'une procédure simple)
- Rigueur (l'ordinateur ne travaille pas *au feeling*)
- Exhaustivité (pouvoir résoudre toutes les grilles)
- Rapidité (aussi optimisé que possible)

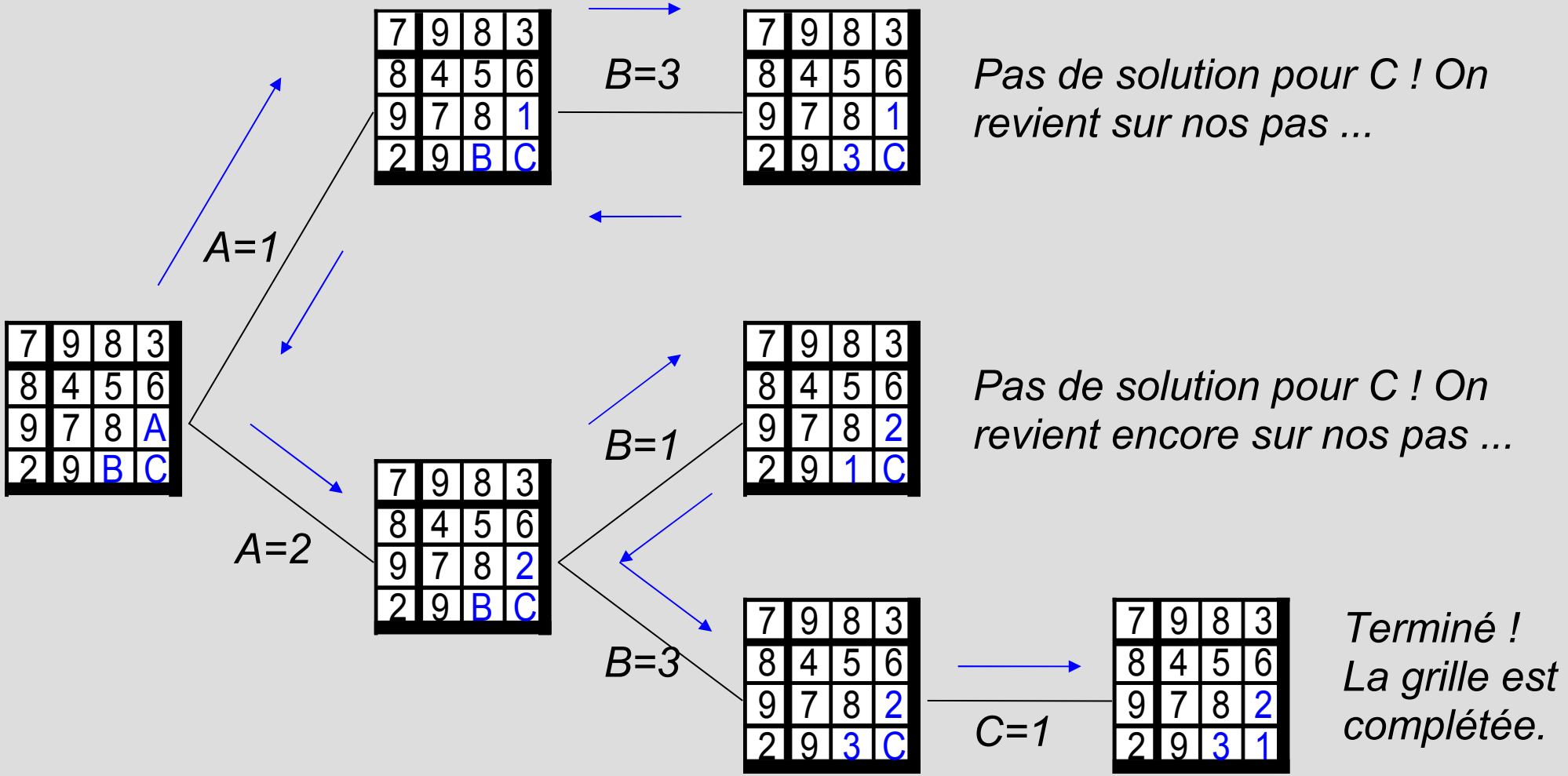
L'algorithme de *sudokusc (1)*

Un algorithme récursif



L'algorithme de *sudokusc (2)*

Exploration de l'arbre



Ici, on a été malchanceux : la solution était tout en bas de l'arbre (on aurait pu tomber dessus dès le début !)

La programmation

C'est quoi ?

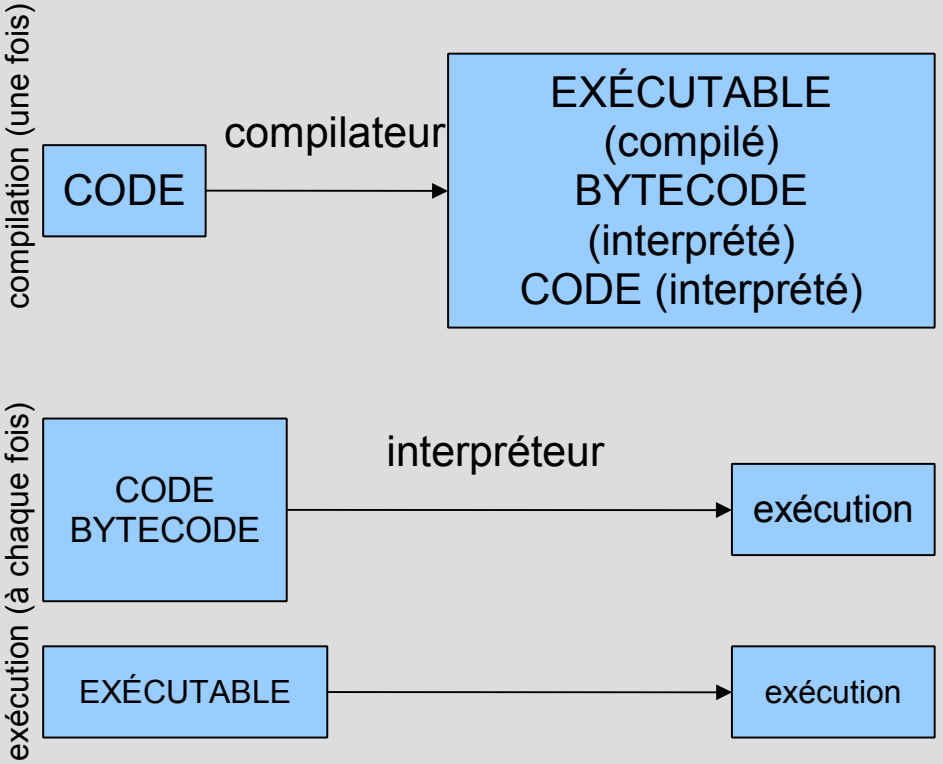
```
#define G(n) int n(int t, int q, int d)
#define X(p,t,s)
(p>=t&&p<(t+s)&&(p-
(t)&1023)<(s&1023))
#define U(m) *((signed char *)(m))
#define F if(!--q){
#define I(s) (int)main-(int)s
#define P(s,c,k) for(h=0;
h>>14==0;
h+=129)Y(16*c+h/1024+Y(V+36))
&128>>(h&7)?U(s+(h&15367))=k:
k

G (B)
{
  Z;
  F D = E (Y (V), C = E (Y (V), Y (t
+ 4) + 3, 4, 0), 2, 0);
  Y (t + 12) = Y (t + 20) = i;
  Y (t + 24) = 1;
  Y (t + 28) = t;
  Y (t + 16) = 442890;
  Y (t + 28) = d = E (Y (V), s = D *
8 + 1664, 1, 0);
  for (p = 0; j < s; j++, p++)
    U (d + j) = i == D | j < p ? p--, 0 :
(n = U (C + 512 + i++)) < ' ' ? p |=
      n * 56 - 497, 0 : n;
}
```

- On rédige l'algorithme dans un langage de programmation
- L'ordinateur traduit ce code en des instructions qu'il peut comprendre et exécuter

Les langages de programmation

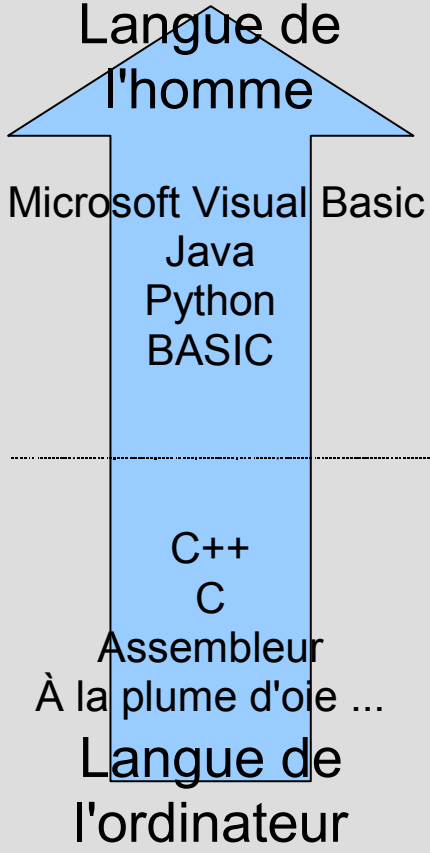
Quel langage choisir ?

INTERPRÉTÉ OU COMPILÉ ?



La compilation doit être refaite si l'on change de machine, mais l'interprétation peut ralentir l'exécution.

HAUT NIVEAU OU BAS NIVEAU ?

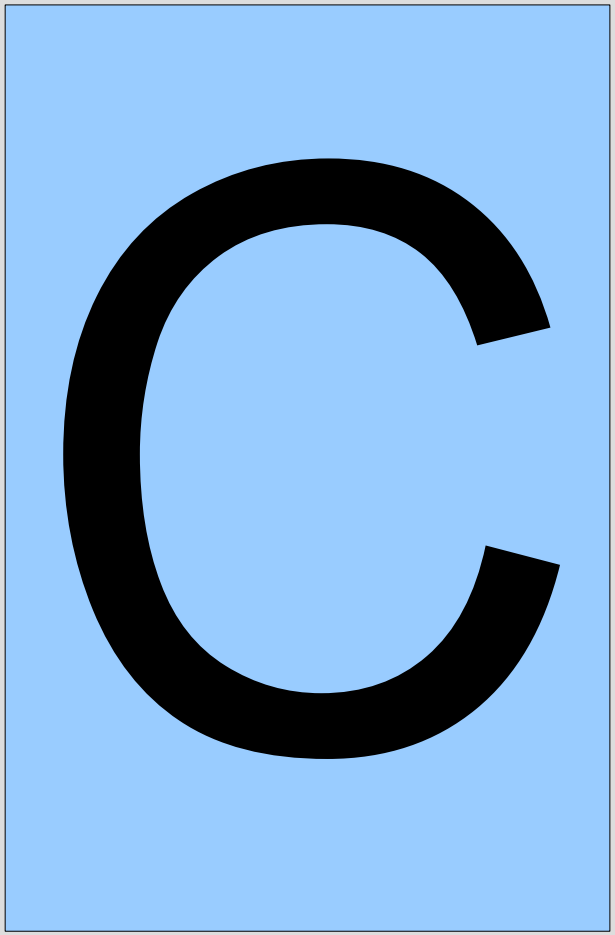


- plus concis (en apparence)
- plus naturel
- plus indépendant de la machine
- moins rapide
- moins rigoureux
- moins de possibilités

- plus de possibilités
- plus rigoureux
- plus rapide
- moins concis (toutefois...)
- moins naturel
- moins indépendant de la machine

Le langage C

C quoi ?



- Bas niveau
- Compilé
- *Très* rapide
- *Très* rigoureux mais permissif (attention !)
- Normalisé ANSI et ISO
- Vieux ! (1972, normalisé en 89 et a évolué depuis)
- Peu de concepts mis en oeuvre
- Quelques « extensions » (C++, etc.)
- Portable (si l'on fait attention à ce que l'on fait)
- Déjà très utilisé
- ...et en plus, je souhaitais l'apprendre...

Les pointeurs

Un concept inventé par le langage C. Mais qu'est-ce-que c'est ?

Mémoire vive de l'ordinateur

Adresses	Valeur
...	...
546023	15
546024	0
546025	
546026	-65
546027	0
546028	185
...	...

La mémoire vive d'un ordinateur est un ensemble d'adresses contenant chacune une valeur. Dans un programme, on utilise des variables pour stocker des données (par exemple, une grille de Sudoku est un tableau de 81 variables). Un pointeur est une variable contenant une adresse (ex : à l'adresse 546025, il y a l'adresse 546027).

Pour optimiser *sudokusc*, on utilise les pointeurs. Par exemple, au lieu de stocker plein de grilles différentes alors que l'on parcourt l'arbre et que l'on essaie de résoudre (ce qui prend beaucoup de place dans la mémoire), on stocke une seule fois la grille et l'on donne aux différents appels du programme (à chaque essai) un pointeur vers la grille, plutôt qu'une copie de la grille.

Le programme

Quel résultat au final ?

```
#include <stdio.h>
#include <stdlib.h>
#include <tgmath.h>
#include <string.h>
#include <stdlib.h>
#define DIM 3
int fill(int grid[DIM*DIM][DIM*DIM], int[2], int[DIM*DIM+1]); int firstempty(int[DIM*DIM][DIM*DIM], int[DIM*DIM*DIM*DIM][2], int *); int solve(int[DIM*DIM][DIM*DIM], int[DIM*DIM*DIM*DIM][2], int); int solution(int[DIM*DIM][DIM*DIM]); int main(int argc, char *argv[]) { int grid[DIM*DIM][DIM*DIM]; int empty[DIM*DIM*DIM*DIM][2]; int empties = 0; printf("Please enter the grid you wish to solve \n"); printf("Separate each digit by enter \n"); printf("Enter 0 for empty squares \n"); int i, j, ret; for(i=0;i<DIM*DIM;i++) { for(j=0;j<DIM*DIM;j++) { scanf("%ld",&grid[i][j]); } printf("- - - - - - - - - \n"); } firstempty(grid, empty, (int *) &empties); if (empties) { solve(grid, empty, empties); }else{ solution(grid); } return 0; } int firstempty(int grid[DIM*DIM][DIM*DIM], int empty[DIM*DIM*DIM*DIM][2], int *empties) { int i, j; for(i=(DIM*DIM)-1;i>=0;i--) { for(j=(DIM*DIM)-1;j>=0;j--) { if (!grid[i][j]) { empty[*empties][0] = i; empty[*empties][1] = j; (*empties)++; } } } return 0; } int solve(int grid[DIM*DIM][DIM*DIM], int empty[DIM*DIM*DIM*DIM][2], int empties) { int i, ret; int fills[DIM*DIM+1] = {0}; empties--; fill(grid, (int *) empty[empties], (int *) &fills); for(i=1;i<DIM*DIM+1;i++) { if (!fills[i]) { grid[empty[empties][0]][empty[empties][1]] = i; if (!empties) { solution(grid); } else { solve(grid, empty, (int) empties); } } } grid[empty[empties][0]][empty[empties][1]] = 0; return 0; } int solution(int grid[DIM*DIM][DIM*DIM]) { int i, j, k; for(i=0;i<DIM*DIM;i++) { for (k=0;k<DIM*DIM;k++) { printf("+-"); } printf("+\n"); for(j=0;j<DIM*DIM;j++) { printf("|%d",grid[i][j]); } printf("|\n"); } for(k=0;k<DIM*DIM;k++) { printf("+-"); } printf("+\n\n"); return 0; } int fill(int grid[DIM*DIM][DIM*DIM], int cell[2], int ret[DIM*DIM+1]) { int k = 0; for(k=0;k<DIM*DIM;k++) { ret[grid[cell[0]][k]] = 1; ret[grid[k][cell[1]]] = 1; ret[grid[(cell[0]/DIM)*DIM+k/DIM][(cell[1]/DIM)*DIM+k%DIM]] = 1; } return 0; }
```

- Environ 280 lignes - 800 mots - 6000 caractères (court !)
- Environ 2000 caractères pour la version courte, à gauche (pas de commentaires ni d'espacement)
- A résolu tous les Sudoku testés
- Temps de résolution : environ 1-2 millisecondes sur ma machine (!!)
- Interface terminal, peu conviviale
- Peut résoudre les Sudoku de dimension *n*
- Logiciel libre sous GPL

Mais pourquoi il a fait ça ?

C'est vrai, ça sert à quoi, son truc ?

Image de David Vignoni, licence GNU LGPL

- Pour apprendre le C
- Pour voir si j'en étais capable
- Au cas où ça n'existait pas encore ...
- Pour m'occuper
- Pour m'amuser, aussi (sisi, ça m'*amuse*)
- Pour ricaner quand je vois quelqu'un en difficulté devant un Sudoku
- Pour réfléchir un peu
- Pour résoudre le cas général
- Au cas où ça viendrait un jour à servir à quelque chose
- Pour le bien de notre civilisation
- Inutile donc indispensable !
- ...

Il vaut mieux pomper même s'il ne se passe rien que risquer qu'il se passe quelque chose de pire en ne pompant pas – Devise Shadok