UNIVERSITÉ
CÔTE D'AZUR

Outils d'aide à la mise au point & Préoccupations énergétiques

Présentation: **Stéphane Lavirotte**
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Stéphane Lavirotte, Sylvain Ferrand

Mail: Stephane.Lavirotte@univ-cotedazur.fr
Web: http://stephane.lavirotte.com/
Université Côte d'Azur

UNIVERSITÉ
CÔTE D'AZUR

Des outils pour la mise au point

De vos programmes C

UNIVERSITÉ
CÔTE D'AZUR

Remarques générales

- ✓ Le temps de débogage d'un programme peut représenter jusqu'à 80% du temps de développement
  - Mais on peut aussi réfléchir avant de coder
- ✓ Bonnes pratiques
  - Convention de nommage
    - Eviter les noms trop proches les uns des autres
    - Utiliser des noms courts et explicites
  - Indenter votre code (automatiquement avec l'IDE)
  - Respecter le style de codage et les conventions d'un langage
  - Commentez le code (mais ne pas le paraphraser)
- ✓ Mais il existe aussi des outils pour vous aider à la mise au point du code

UNIVERSITÉ CÔTE D'AZUR

Qu'est ce qu'un débogueur ?

- ✓ Destiné à aider le programmeur à détecter ses bugs
- ✓ Permet en général:
  - Exécuter un programme pas à pas
  - Contrôler l'état de la mémoire et des variables
  - Agir sur la mémoire
  - Définir des point d'arrêts
  - ...
- ✓ Plusieurs débogueurs disponibles
  - Sous Windows: Visual Studio Debugger (Microsoft)
  - Sous Unix et Linux: gdb (GNU), dbx (IBM), ddd (GNU)
  - Mais aussi débogueur de mémoire (fuite mémoire par exemple)
    - Purify (Unicom systems), Valgrind

UNIVERSITÉ
CÔTE D'AZUR

GDB: GNU Debugger

- ✓ Le débogueur standard de GNU
  - Fonctionne sur un grand nombre d'Unix et d'architectures
  - Logiciel en ligne de commande
- ✓ Deux modes d'utilisation
  - Débogage d'un programme en cours d'exécution
  - Débogage post-mortem (fichier core)
- ✓ Nécessité de compiler avec l'option –g
  - Inclus les symboles de débogage
- ✓ Débogueur de « bas niveau »
  - Interaction via des commandes
  - Pas de « vrai » retour au source
  - Souvent appelé par d'autres programmes
    - Eclipse, DDD, Visual Code

UNIVERSITÉ
CÔTE D'AZUR

Valgrind

- ✓ Ensemble d'outils pour le débogage de programmes
- ✓ Principe de fonctionnement
  - Exécute le programme dans une machine virtuelle
  - Analyse le fonctionnement en cours d'exécution
- ✓ **Memcheck** vérifie que l'on:
  - N'utilise pas de valeurs ou pointeurs non initialisés
    - Peut-être vérifié par le compilateur
  - Ne lit pas de zones mémoires libérées
    - Comportement aléatoire si la zone est réécrite ou pas
  - Ne lit pas de zones mémoires en dehors de ce que l'on a alloué
    - Comportement aléatoire en fonction des données dans la zone
  - N'oublie pas de libérer la mémoire
    - Eviter les fuites mémoires (programme qui utilise toujours plus de mémoire au cours de son exécution)

UNIVERSITÉ CÔTE D'AZUR

Analyse d'un « buffer overflow »

- ✓ Le dépassement d'une zone mémoire est un bug très classique
  - Peut avoir des conséquences graves pour la sécurité
  - Permet par exemple d'exécuter du code avec les droits de l'utilisateur
- ✓ La pile est une partie de la mémoire disponible
  - Pour stocker les variables locales d'un programme (vue précédemment)
- ✓ Si une variable déborde sur la pile, alors possible de:
  - Modifier d'autres variables
  - Modifier l'adresse de retour d'une fonction
    - Et donc de faire exécuter un autre code…
  - Déclenche certaines fois une erreur, mais pas toujours

UNIVERSITÉ CÔTE D'AZUR

Stack Buffer Overflow 1/3

- ✓ Exemple de code apparemment anodin

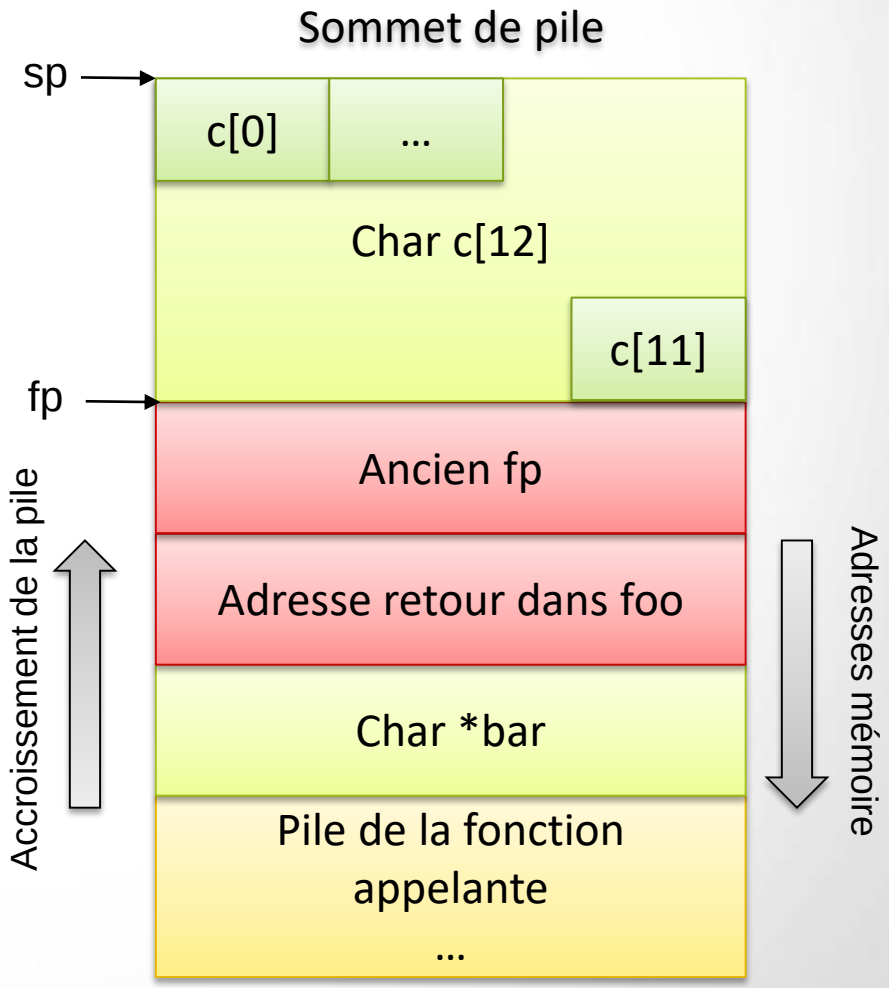
```
void foo(char *bar) {
    char c[12];
    strcpy(c, bar); // pas de vérification de taille
}
int main(int argc, char **argv)
{
    foo(argv[1]);
}
```

- ✓ Mais ce code présente un danger d'overflow
- ✓ Remarque on ne vérifie pas la taille de la chaîne prise en argument du programme

UNIVERSITÉ CÔTE D'AZUR

Stack Buffer Overflow 2/3

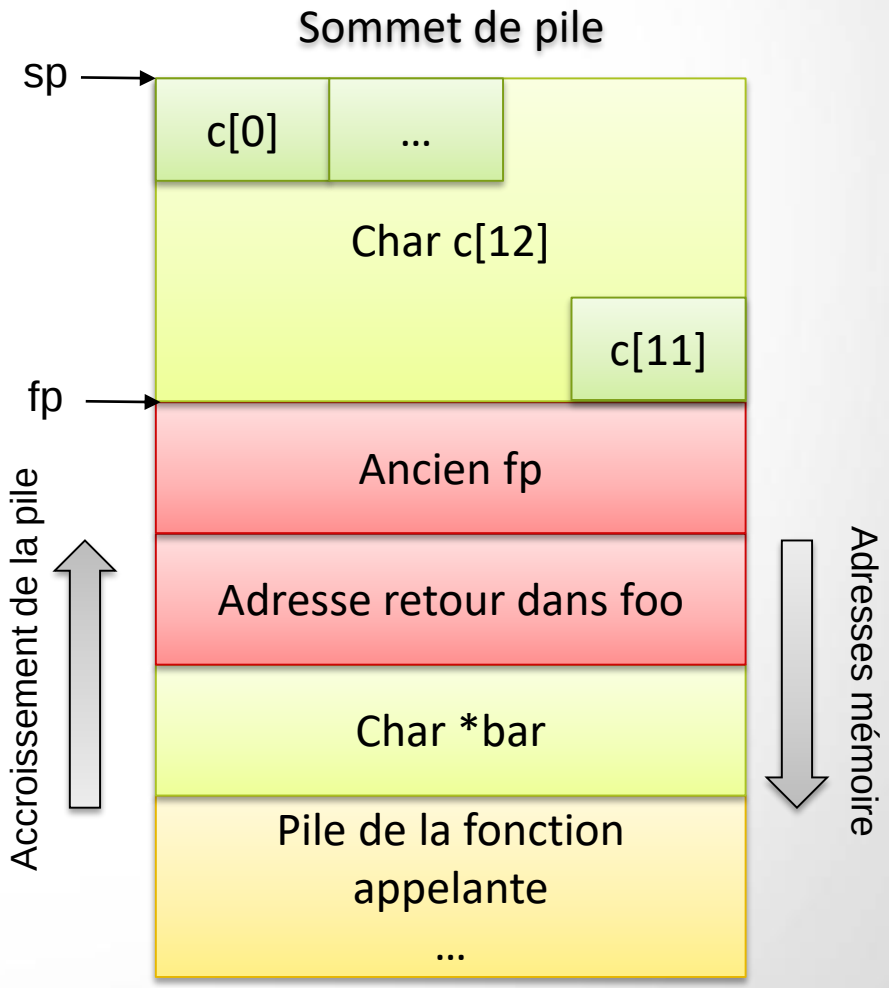
- ✓ Voici la structure de la pile lors de l'exécution de `foo`
- ✓ Le système met dans la pile:
  - Les arguments d'appel de la fonction
  - L'adresse de retour vers a fonction appelante
  - L'adresse de début
  - Les variables locales



UNIVERSITÉ CÔTE D'AZUR

Stack Buffer Overflow 3/3

- ✓ Exécution avec différents arguments
- ✓ Si la chaîne passée en argument est <= 11 caractères, pas de souci
- ✓ Si la chaine passée en argument à une taille supérieur, cela peut alors écraser
  - D'éventuelles variables
  - Les arguments de la fct
  - Voir l'adresse de retour



UNIVERSITÉ
CÔTE D'AZUR

Sensibilisation au Numérique Responsable

Une brève introduction… qui justifie bien l'apprentissage de C !

UNIVERSITÉ CÔTE D'AZUR

Les équipements: limiter le renouvellement !

- ✓ Ademe, « En route vers la sobriété numérique » (2022)
  - 70 à 80% de l'empreinte carbone du numérique en France est due à la fabrication des appareils
    - Vecteur principal sur lequel faire preuve de sobriété
- ✓ Sur quels points agir :
  - Faire évoluer son matériel pour éviter un rachat non essentiel
  - Ne renouveler ses appareils que lorsque définitivement HS
  - S'interroger sur ses besoins réels
  - Privilégier l'achat d'équipements avec un indice de réparabilité élevé
  - Penser aux équipements reconditionnés
  - Ne pas jeter mais recycler: un vieil appareil peut être un gisement de matière premières
  - Entretenez vos appareils (nettoyage, durée de vie batterie, ...)

UNIVERSITÉ CÔTE D'AZUR

Programmer: oui avec quel langage ?

- ✓ Classification des langages suivant les paradigmes:
  - Impératif, Orienté Objet, Fonctionnel, ...
- ✓ Classification des langages en 2 grandes catégories:
  - Compilé: C, C++, Rust, Go, ...
  - Interprété: Javascript, Perl, Python, Ruby, PHP, ...
  - Mais aussi compilé et interprété: Java, C#, ...

- ✓ Etude de R. Pereira etal. « *Energy accross programming languages: how do energy, time and memory relate?* »
  - « *Peut-on comparer l'efficacité énergétique des langages de programmation ?* »,
    - « *Le langage le plus rapide est-il toujours le plus efficace énergétiquement ?* »,
    - « *Quel est le rapport entre l'utilisation de la mémoire et la consommation d'énergie ?* »,
    - « *Pouvons-nous décider automatiquement quel est le meilleur langage de programmation en tenant compte de l'énergie, du temps et de l'utilisation de la mémoire ?* »

UNIVERSITÉ CÔTE D'AZUR

Langage le plus rapide, le plus efficace énergétiquement ? 1/3

- ✓ Energy (J) = Power (W) x Time(s)

binary-trees	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust ⇓$_2$	49.07	1263	0.039	180
(c) Fortran ⇑$_1$	69.82	2112	0.033	133
(c) Ada ⇓$_1$	95.02	2822	0.034	197
(c) Ocaml ↓$_1$ ⇑$_2$	100.74	3525	0.029	148
(v) Java ↑$_1$ ⇓$_{16}$	111.84	3306	0.034	1120
(v) Lisp ↓$_3$ ⇓$_3$	149.55	10570	0.014	373
(v) Racket ↓$_4$ ⇓$_6$	155.81	11261	0.014	467
(i) Hack ↑$_2$ ⇓$_9$	156.71	4497	0.035	502
(v) C# ↓$_1$ ⇓$_1$	189.74	10797	0.018	427
(v) F# ↓$_3$ ⇓$_1$	207.13	15637	0.013	432
(c) Pascal ↓$_3$ ⇑$_5$	214.64	16079	0.013	256
(c) Chapel ↑$_5$ ⇑$_4$	237.29	7265	0.033	335
(v) Erlang ↑$_5$ ⇑$_1$	266.14	7327	0.036	433
(c) Haskell ↑$_2$ ⇓$_2$	270.15	11582	0.023	494
(i) Dart ↓$_1$ ⇑$_1$	290.27	17197	0.017	475
(i) JavaScript ↓$_2$ ⇓$_4$	312.14	21349	0.015	916
(i) TypeScript ↓$_2$ ⇓$_2$	315.10	21686	0.015	915
(c) Go ↑$_3$ ⇑$_{13}$	636.71	16292	0.039	228
(i) Jruby ↑$_2$ ⇓$_3$	720.53	19276	0.037	1671
(i) Ruby ⇑$_5$	855.12	26634	0.032	482
(i) PHP ⇑$_3$	1,397.51	42316	0.033	786
(i) Python ⇑$_{15}$	1,793.46	45003	0.040	275
(i) Lua ↓$_1$	2,452.04	209217	0.012	1961
(i) Perl ↑$_1$	3,542.20	96097	0.037	2148
(c) Swift	n.e.			

fannkuch-redux	Energy	Time	Ratio	Mb
(c) C ⇓$_2$	215.92	6076	0.036	2
(c) C++ ⇑$_1$	219.89	6123	0.036	1
(c) Rust ⇓$_{11}$	238.30	6628	0.036	16
(c) Swift ⇓$_5$	243.81	6712	0.036	7
(c) Ada ⇓$_2$	264.98	7351	0.036	4
(c) Ocaml ↓$_1$	277.27	7895	0.035	3
(c) Chapel ↑$_1$ ⇓$_{18}$	285.39	7853	0.036	53
(v) Lisp ↓$_3$ ⇓$_{15}$	309.02	9154	0.034	43
(v) Java ↑$_1$ ⇓$_{13}$	311.38	8241	0.038	35
(c) Fortran ⇓$_1$	316.50	8665	0.037	12
(c) Go ↑$_2$ ⇑$_7$	318.51	8487	0.038	2
(c) Pascal ⇑$_{10}$	343.55	9807	0.035	2
(v) F# ↓$_1$ ⇓$_7$	395.03	10950	0.036	34
(v) C# ↑$_1$ ⇓$_5$	399.33	10840	0.037	29
(i) JavaScript ↓$_1$ ⇓$_2$	413.90	33663	0.012	26
(c) Haskell ↑$_1$ ⇑$_8$	433.68	14666	0.030	7
(i) Dart ⇓$_7$	487.29	38678	0.013	46
(v) Racket ⇑$_3$	1,941.53	43680	0.044	18
(v) Erlang ⇑$_3$	4,148.38	101839	0.041	18
(i) Hack ⇓$_6$	5,286.77	115490	0.046	119
(i) PHP	5,731.88	125975	0.046	34
(i) TypeScript ↓$_4$ ⇑$_4$	6,898.48	516541	0.013	26
(i) Jruby ↑$_1$ ⇓$_4$	7,819.03	219148	0.036	669
(i) Lua ↓$_3$ ⇑$_{19}$	8,277.87	635023	0.013	2
(i) Perl ↑$_2$ ⇑$_{12}$	11,133.49	249418	0.045	12
(i) Python ↑$_2$ ⇑$_{14}$	12,784.09	279544	0.046	12
(i) Ruby ↑$_2$ ⇑$_{17}$	14,064.98	315583	0.045	8

fasta	Energy	Time	Ratio	Mb
(c) Rust ⇓$_9$	26.15	931	0.028	16
(c) Fortran ↓$_6$	27.62	1661	0.017	1
(c) C ↑$_1$ ⇓$_1$	27.64	973	0.028	3
(c) C++ ↑$_1$ ⇓$_2$	34.88	1164	0.030	4
(v) Java ↑$_1$ ⇓$_{12}$	35.86	1249	0.029	41
(c) Swift ⇓$_9$	37.06	1405	0.026	31
(c) Go ↓$_2$	40.45	1838	0.022	4
(c) Ada ↓$_2$ ⇑$_3$	40.45	2765	0.015	3
(c) Ocaml ↓$_2$ ⇓$_{15}$	40.78	3171	0.013	201
(c) Chapel ↑$_5$ ⇓$_{10}$	40.88	1379	0.030	53
(v) C# ↑$_4$ ⇓$_5$	45.35	1549	0.029	35
(i) Dart ⇓$_6$	63.61	4787	0.013	49
(i) JavaScript ⇓$_1$	64.84	5098	0.013	30
(c) Pascal ↓$_1$ ⇑$_{13}$	68.63	5478	0.013	0
(i) TypeScript ↓$_2$ ⇓$_{10}$	82.72	6909	0.012	271
(v) F# ↑$_2$ ⇑$_3$	93.11	5360	0.017	27
(v) Racket ↓$_1$ ⇑$_5$	120.90	8255	0.015	21
(c) Haskell ↑$_2$ ⇓$_8$	205.52	5728	0.036	446
(v) Lisp ⇓$_2$	231.49	15763	0.015	75
(i) Hack ⇓$_3$	237.70	17203	0.014	120
(i) Lua ⇑$_{18}$	347.37	24617	0.014	3
(i) PHP ↓$_1$ ⇑$_{13}$	430.73	29508	0.015	14
(v) Erlang ↑$_1$ ⇑$_{12}$	477.81	27852	0.017	18
(i) Ruby ↓$_1$ ⇑$_2$	852.30	61216	0.014	104
(i) JRuby ↑$_1$ ⇓$_2$	912.93	49509	0.018	705
(i) Python ↓$_1$ ⇑$_{18}$	1,061.41	74111	0.014	9
(i) Perl ↑$_1$ ⇑$_8$	2,684.33	61463	0.044	53

UNIVERSITÉ CÔTE D'AZUR

Langage le plus rapide, le plus efficace énergétiquement ? 1/3

- ✓ Sur le benchmark précédent (3 algorithmes sur les 13 étudiés):
  - Binary-trees : Allouer, parcourir et désallouer de nombreux arbres binaires
  - Fannkuch-redux: Accès indexé à de petites séquences d'entiers
  - Fasta: Générer et écrire des séquences d'ADN
- ✓ Les langages consommant le moins d'énergie:
  - C, C++, Rust sur les 2 premiers et Rust, Fortan et C pour fasta
  - Tous ces langages sont compilés
- ✓ Les langages consommant le plus d'énergie:
  - Perl, Python, Lua et Ruby/JRuby
  - Tous ces langages sont interprétés
- ✓ Mais, non, un langage plus rapide n'est pas toujours le plus économe en énergie

UNIVERSITÉ CÔTE D'AZUR

Langage le plus rapide, le plus efficace énergétiquement ? 1/3

- ✓ Fortan / Fasta: plus long que d'autres langages, mais moins énergivore (donc moins de puissance utilisée)

binary-trees	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust ⇓2	49.07	1263	0.039	180
(c) Fortran ⇑1	69.82	2112	0.033	133
(c) Ada ⇓1	95.02	2822	0.034	197
(c) Ocaml ↓1 ⇑2	100.74	3525	0.029	148
(v) Java ↑1 ⇓16	111.84	3306	0.034	1120
(v) Lisp ↓3 ⇓3	149.55	10570	0.014	373
(v) Racket ↓4 ⇓6	155.81	11261	0.014	467
(i) Hack ↑2 ⇓9	156.71	4497	0.035	502
(v) C# ↓1 ⇓1	189.74	10797	0.018	427
(v) F# ↓3 ⇓1	207.13	15637	0.013	432
(c) Pascal ↓3 ⇑5	214.64	16079	0.013	256
(c) Chapel ↑5 ⇑4	237.29	7265	0.033	335
(v) Erlang ↑5 ⇑1	266.14	7327	0.036	433
(c) Haskell ↑2 ⇓2	270.15	11582	0.023	494
(i) Dart ↓1 ⇑1	290.27	17197	0.017	475
(i) JavaScript ↓2 ⇓4	312.14	21349	0.015	916
(i) TypeScript ↓2 ⇓2	315.10	21686	0.015	915
(c) Go ↑3 ⇑13	636.71	16292	0.039	228
(i) Jruby ↑2 ⇓3	720.53	19276	0.037	1671
(i) Ruby ⇑5	855.12	26634	0.032	482
(i) PHP ⇑3	1,397.51	42316	0.033	786
(i) Python ⇑15	1,793.46	45003	0.040	275
(i) Lua ↓1	2,452.04	209217	0.012	1961
(i) Perl ↑1	3,542.20	96097	0.037	2148
(c) Swift	n.e.			

fannkuch-redux	Energy	Time	Ratio	Mb
(c) C ⇓2	215.92	6076	0.036	2
(c) C++ ⇑1	219.89	6123	0.036	1
(c) Rust ⇓11	238.30	6628	0.036	16
(c) Swift ⇓5	243.81	6712	0.036	7
(c) Ada ⇓2	264.98	7351	0.036	4
(c) Ocaml ↓1	277.27	7895	0.035	3
(c) Chapel ↑1 ⇓18	285.39	7853	0.036	53
(v) Lisp ↓3 ⇓15	309.02	9154	0.034	43
(v) Java ↑1 ⇓13	311.38	8241	0.038	35
(c) Fortran ⇓1	316.50	8665	0.037	12
(c) Go ↑2 ⇑7	318.51	8487	0.038	2
(c) Pascal ⇑10	343.55	9807	0.035	2
(v) F# ↓1 ⇓7	395.03	10950	0.036	34
(v) C# ↑1 ⇓5	399.33	10840	0.037	29
(i) JavaScript ↓1 ⇓2	413.90	33663	0.012	26
(c) Haskell ↑1 ⇑8	433.68	14666	0.030	7
(i) Dart ⇓7	487.29	38678	0.013	46
(v) Racket ⇑3	1,941.53	43680	0.044	18
(v) Erlang ⇑3	4,148.38	101839	0.041	18
(i) Hack ⇓6	5,286.77	115490	0.046	119
(i) PHP	5,731.88	125975	0.046	34
(i) TypeScript ↓4 ⇑4	6,898.48	516541	0.013	26
(i) Jruby ↑1 ⇓4	7,819.03	219148	0.036	669
(i) Lua ↓3 ⇑19	8,277.87	635023	0.013	2
(i) Perl ↑2 ⇑12	11,133.49	249418	0.045	12
(i) Python ↑2 ⇑14	12,784.09	279544	0.046	12
(i) Ruby ↑2 ⇑17	14,064.98	315583	0.045	8

fasta	Energy	Time	Ratio	Mb
(c) Rust ⇓9	26.15	931	0.028	16
(c) Fortran ↓6	27.62	1661	0.017	1
(c) C ↑1 ⇓1	27.64	973	0.028	3
(c) C++ ↑1 ⇓2	34.88	1164	0.030	4
(v) Java ↑1 ⇓12	35.86	1249	0.029	41
(c) Swift ⇓9	37.06	1405	0.026	31
(c) Go ↓2	40.45	1838	0.022	4
(c) Ada ↓2 ⇑3	40.45	2765	0.015	3
(c) Ocaml ↓2 ⇓15	40.78	3171	0.013	201
(c) Chapel ↑5 ⇓10	40.88	1379	0.030	53
(v) C# ↑4 ⇓5	45.35	1549	0.029	35
(i) Dart ⇓6	63.61	4787	0.013	49
(i) JavaScript ⇓1	64.84	5098	0.013	30
(c) Pascal ↓1 ⇑13	68.63	5478	0.013	0
(i) TypeScript ↓2 ⇓10	82.72	6909	0.012	271
(v) F# ↑2 ⇑3	93.11	5360	0.017	27
(v) Racket ↓1 ⇑5	120.90	8255	0.015	21
(c) Haskell ↑2 ⇓8	205.52	5728	0.036	446
(v) Lisp ⇓2	231.49	15763	0.015	75
(i) Hack ⇓3	237.70	17203	0.014	120
(i) Lua ⇑18	347.37	24617	0.014	3
(i) PHP ↓1 ⇑13	430.73	29508	0.015	14
(v) Erlang ↑1 ⇑12	477.81	27852	0.017	18
(i) Ruby ↓1 ⇑2	852.30	61216	0.014	104
(i) JRuby ↑1 ⇓2	912.93	49509	0.018	705
(i) Python ↓1 ⇑18	1,061.41	74111	0.014	9
(i) Perl ↑1 ⇑8	2,684.33	61463	0.044	53

UNIVERSITÉ CÔTE D'AZUR

Quel rapport entre mémoire et consommation d'énergie ?

- ✓ Les moins gourmands en mémoire:
  - Pascal, Go, C, Fortan C++
- ✓ Les plus gourmands en mémoire:
  - Jruby, Dart, Erlang, Lua, Perl et ... java
- ✓ Consommation d'énergie DRAM à peut de relation avec la quantité de mémoire à un instant (peut être plus avec la manière dont elle est utilisée)

	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

UNIVERSITÉ CÔTE D'AZUR

Energie & Temps & Mémoire

- ✓ Ensembles Pareto optimaux pour différentes combinaisons d'objectifs

Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go Rust • C++ • Fortran	C Rust	C • Pascal Rust • C++ • Fortran • Go	C • Pascal • Go Rust • C++ • Fortran
Ada Java • Chapel • Lisp • Ocaml Haskell • C# Swift • PHP F# • Racket • Hack • Python JavaScript • Ruby	C++ Ada Java Pascal • Chapel Lisp • Ocaml • Go Fortran • Haskell • C# Swift Dart • F# JavaScript Racket TypeScript • Hack PHP	Ada Java • Chapel • Lisp OCaml • Swift • Haskell C# • PHP Dart • F# • Racket • Hack • Python JavaScript • Ruby	Ada Java • Chapel • Lisp • Ocaml Swift • Haskell • C# Dart • F# • Racket • Hack • PHP JavaScript • Ruby • Python TypeScript • Erlang
Dart • TypeScript • Erlang JRuby • Perl Lua	Erlang Lua • JRuby Ruby	TypeScript Erlang • Lua • Perl JRuby	Lua • JRuby • Perl

UNIVERSITÉ CÔTE D'AZUR

Conclusion

- ✓ Pouvons-nous décider automatiquement quel est le meilleur langage logiciel en tenant compte de l'énergie, du temps et de l'utilisation de la mémoire ?
  - Si on ne considère que Energie et Temps alors oui:
    - C'est le langage C (avec Rust et C++ qui sont proches)
  - Si on considère aussi l'utilisation mémoire...
    - Il faut faire entrer d'autres paramètres en compte
      - Fonctionnel ou pas, Objet ou pas, ...
- ✓ Réfléchissez aussi à d'autres paramètres
  - Facilité d'écriture dans un langage
  - Bibliothèques disponibles
  - Productivité
  - Exécution courte et ponctuelle ou exécution permanente
- ✓ Bref, pas un choix car le langage est à la mode !
  - Choix basés sur de vrais critères scientifiques !

UNIVERSITÉ CÔTE D'AZUR

Programmer de manière optimale reste primordial

- ✓ Même si l'on choisi le bon langage
- ✓ Il faut programmer un traitement optimal des données
- ✓ Complexité des algorithmes
  - Algorithme de tri stupide

```
fonction tri_stupide(liste)
    tant que la liste n'est pas triée
        mélanger aléatoirement les éléments de la liste
```

    - Complexité dans le pire cas: ∞
    - Complexité dans le cas moyen: $(n+1)!$
  - Algorithme tri rapide (ou *quicksort*)
    - Complexité dans le pire cas: n^2
    - Complexité dans le cas moyen: $n \log(n)$
- ✓ C'est pourquoi, nous avons souvent insisté durant ce cours sur la production de bon code !
  - Il vaut aussi mieux faire un algo en n que en $2n$

UNIVERSITÉ CÔTE D'AZUR

Mais bien d'autres axes d'amélioration énergétique

- ✓ Hors du scope de ce cours:
  - Transport des données
    - Technologies utilisées (WiFi vs 4G, 4G vs 5G)
    - Localisation des traitements et des stockages
  - Eco-conception des sites Web
    - Tester des pages de sites avec l'Ecoindex:
      - https://www.ecoindex.fr/
  - Pratiques courantes:
    - Rédaction des emails en texte plutôt qu'en HTML
    - Utilisation des réseaux sociaux (connexion permanente)
    - Utilisation des vidéos (60% du trafic mondial)
    - Résolution des vidéos demandées (résolution x2 => 4x plus de données)
- ✓ En tant que futur ingénieur en informatique c'est aussi de votre responsabilité dans vos créations de demain