Le langage ZIG

Victor MALOD - Yaël PARA

Introduction

Zig est un **langage de programmation** à usage général, il a été conçu pour écrire des programmes :

- **Maintenables**
- **Robustes**
- **Optimaux**
- **Réutilisables**

Langages similaires

Introduction

Le développement de Zig est financé par la **Zig Software Foundation**.

Président : **Andrew Kelley**
Équipe : **employés à temps plein**
Financements : **donations**

Créé par
Andrew Kelley
en **2016**

Introduction

Présentation :

1. Objectifs de Zig
2. Les caractéristiques du langage
3. Performances et comparaisons
4. Démonstration

Objectifs du langage

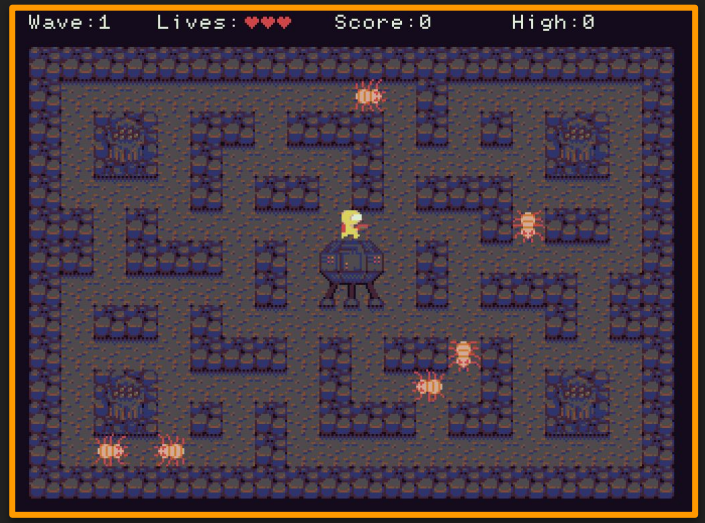
Fournir une **version améliorée** de C

Langage **minimaliste**

Tout en ayant les features de ses concurrents

Exemple de projet :

le jeu **Oxid**



Les caractéristiques de Zig

Zig possèdes quelques caractéristiques de base :

- Aucun flux de contrôle "caché"
- Pas de pré-processing, ni de macro
- Interopérabilité par conception
- Exécution de code à la compilation
- Gestion de la mémoire uniquement à la charge du développeur
- Sécurité d'exécution réglable

Les caractéristiques de Zig

1. Aucun flux de contrôle caché

Pas de **property function**

Pas d'**overloading** d'opérateur

Pas d'exception **throw/catch**

```
var a = b + c.d;
foo();
bar();
```

Permet d'améliorer la **lisibilité** du code

Les caractéristiques de Zig

2. Pas de pré-processing

Désavantages du **préprocesseur C** :

- Langages **différents**
- **#include** coûteux en temps de compilation
- Erreurs de syntaxe non détectées avec **#ifdef**

```
#ifdef CONST
foo();
#endif
```

Les caractéristiques de Zig

2. Pas de pré-processing

Changements dans **le langage Zig** :

- Un **seul** langage
- Donc **erreurs détectables**
- Fournit les features utiles du préprocesseur C

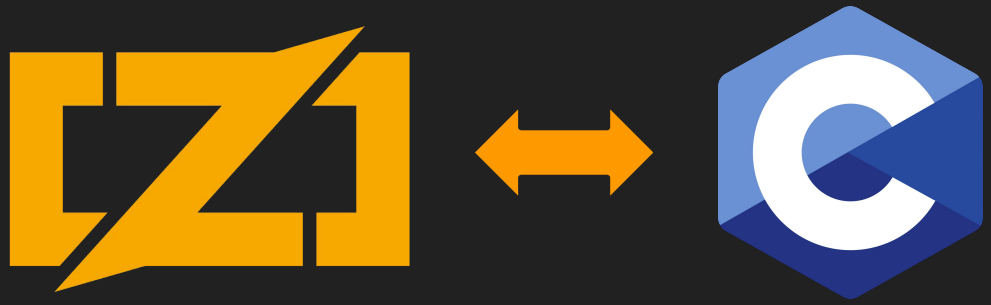
```
const opts =
@import("build_options");


if (opts.CONST)
        foo();
else
        ...
```

Les caractéristiques de Zig

3. Interopérable dans sa conception

Interopérabilité : fonctionner ensemble et partager des données.



Les caractéristiques de Zig

4. Exécution de code à la compilation

On peut demander à Zig d'évaluer une variable, une fonction ou une portion de code à la compilation.

Grâce au mot clé `comptime`.

```
fn multiply(a: i64, b: i64) i64 {
    return a * b;
}

pub fn main() void {
    const len = comptime multiply(4, 5);
    const my_static_array: [len]u8 = undefined;
}
```

Performances

Compilation

Langage \ Test	Temps	Commande
Zig	10.1 s	zig build-exe test.zig
C	5.2 s	gcc test.c
C++	1 min 25 s	g++ test.cpp

Comparaison du temps de compilation pour un fichier source de 400 000 lignes

source : https://vlang.io/compilation_speed

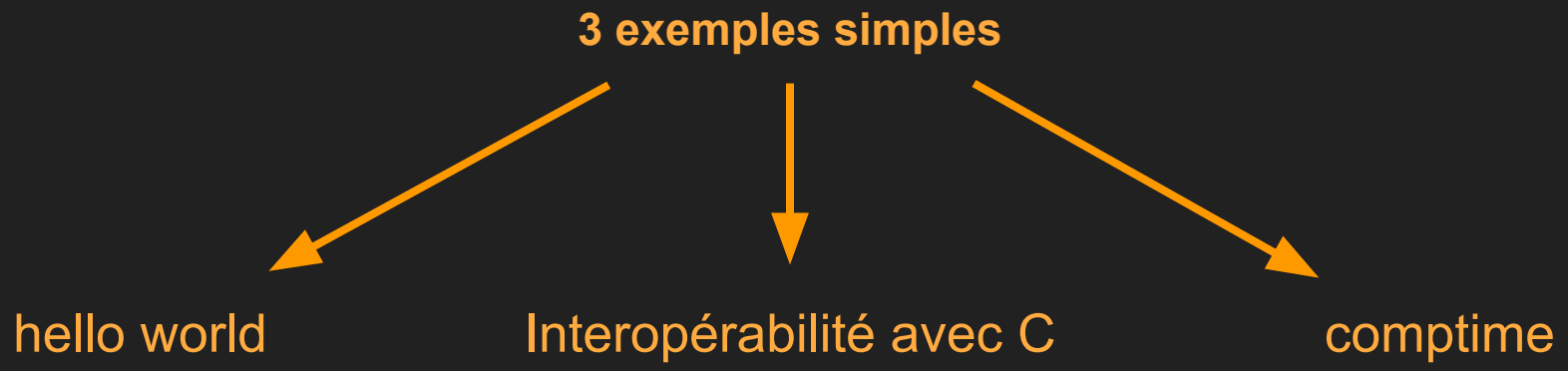
Performances

Exécution

Langage \ Test	Hello world	nbody (5M)	nsieve (12)
Zig	3.4 ms	316 ms	**449 ms**
C	2.4 ms	394 ms	1036 ms
C++	**2.0 ms**	**225 ms**	1015 ms
Rust	2.6 ms	355 ms	780 ms

source : https://programming-language-benchmarks.vercel.app/

Démonstration



Conclusion

Que penser de Zig ?

Offre beaucoup de possibilités similaires à d'autres langages

Performance et lisibilité du code mises en avant

Langage encore récent

Sources

Oxid : https://github.com/dbandstra/oxid

Documentation & vue d'ensemble de Zig : https://ziglang.org/learn/overview/

Projets Zig : https://github-wiki-see.page/m/ziglang/zig/wiki/Community-Projects

Préprocesseur & Macros :
https://andrewkelley.me/post/intro-to-zig.html#preprocessor-alternatives

https://stackoverflow.com/questions/35630480/clang-parser-ignore-directive-ifdef-parse-everything

Revue : https://hackaday.com/2021/10/05/need-a-new-programming-language-try-zig/