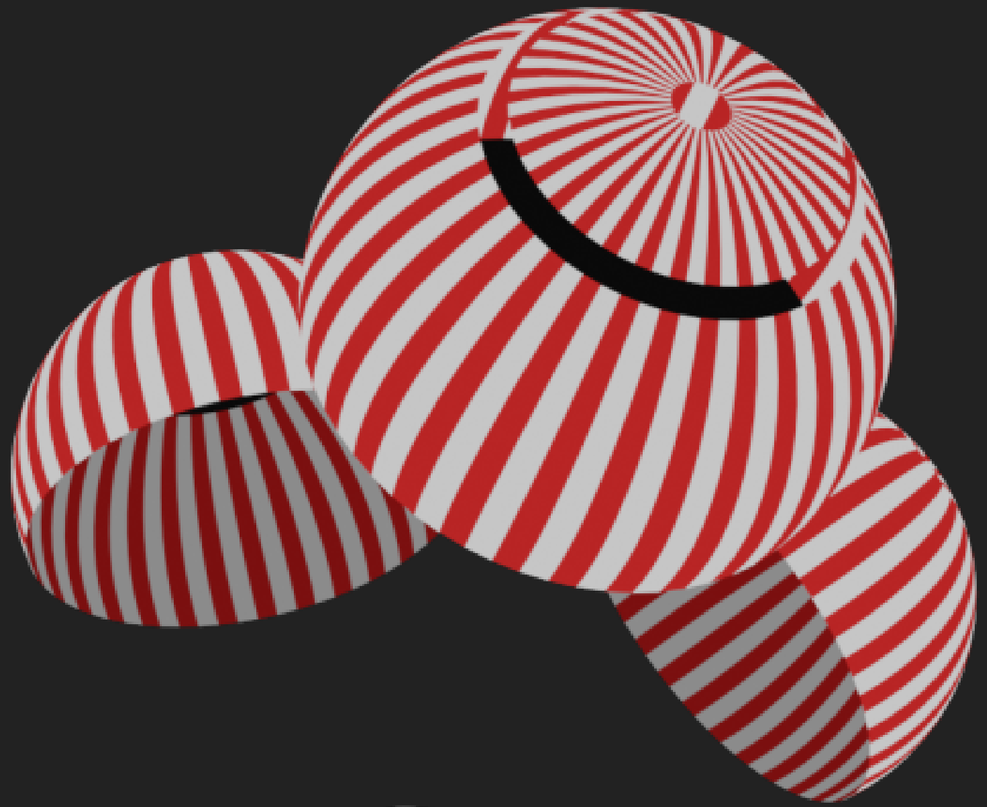

Apollo

arthur.aillet@epitech.eu
julian.scott@epitech.eu
auguste.frater@epitech.eu
ludovic.de-chavagnac@epitech.eu

EPITECH.
L'ECOLE DE L'EXPERTISE INFORMATIQUE

Apollo

01

Apollo est notre proposition de langage de programation suivant le projet **GLaDOS**. Nous avons cherché à faire un **langage de scripting** simple à utilisé comme le bash mais à la syntaxe plus claire et accessible.

Impératif

Le language est **impératif**, il a des **variables mutables** (qui peuvent changer de valeur mais non de type).

Il y a deux keywords utilisables pour créer des boucles: le **while** et le **for**

Impératif

Variables mutables

while (condition)

for i in itérable

Bases opératoires

04

Apollo a des **types** standard et permet de faire des **opérations** dessus facilement grâce à des opérateurs.

Bases opératoires

- Int
- Float
- Bool
- Char
- Array
- String

+
-
*
/
%
==
<
<=
>
>=
!=
&&
||

Typage statique

De plus, **Apollo** est entièrement **statiquement typé** avec une détection complète des erreurs de type au moment de la compilation.
Pour convertir un type en un autre il existe l'operateur "**as**".

Typage statique

07

‘a’ as int

4 as float

0.0 as bool

...

Différent types de fonctions

Apollo dispose de 3 catégories de fonctions:

- **appels shell**, précédées d'un "**$**"
- écrites par l'**utilisateur**, précédées d'un "**@**"
- **natives** a **Apollo**, sans préfixe

Différent types de fonctions

09

Fonctions:

```
@foo(3.141592);
@bar (23, True);
```

Fonctions shell:

```
$echo(“bar”);
$touch (“foo.txt”);
(...)
```

Fonctions natives:

```
print(“hello world”);
append(“filename”);
read();
write();
```

Appel Shell

Apollo peut faire appel a n’importe quel **executable** dans le **$PATH**

Appel Shell


```
@main(string fileName) {
    print($cat(fileName));
}
```

Appel aux exécutables de l'ordinateur

- ls
- cd
- alias
- mkdir
- chmod
- exit
- cp
- rm
- man
- mv
- ...

Opérations sur des tableaux

Le type Array permet plusieurs opérations uniques:

retourner l'élément n du tableau

fusioner deux tableaux

retourner la taille du tableau

Opérations sur des tableaux

array [index] → accès élément

array : array2 → concaténation

len (array) → taille

Sucre syntaxique

Apollo facilite l'écriture de fonctions par le biais de **sucre syntaxique**.

Par exemple, le type **String** est un **Array** de **Char**

Sucre syntaxique

a++ → a = a + 1

a += 1 → a = a + 1

string a = “str“ → [char] a = [’s’, ‘t’, ‘r’]

Stack Trace

Lors de la compilation, Apollo gère les erreures de syntaxe et pour chaque erreur, donne la fonction d'où l'erreur proviens, son type et sa position.
Ces erreures remontent jusqu'au scope de la fonction.

Stack Trace

```
@foo(int a, int b, int stop) int {
   return 4
}
@bar(string b) int {
   if ((b[1]) == ‘a’) {
      for i in b {
         return 5
      }
   }
}
```

```
Errors found during parsing:
        in "bar"(3:0): Syntax error: instruction if is not valid started at 4:0 and finished at 4:3
        in "bar"(3:0): Syntax error: instruction for is not valid started at 4:13 and finished at 5:4
        in "bar"(3:0): Syntax error: instruction return is not valid started at 5:14 and finished at 7:0
        in "foo"(0:0): Syntax error: instruction return is not valid started at 1:0 and finished at 2:0

*** Exception: ExitFailure 1
```

Évaluation à la compilation

Lors de la **compilation**, **Apollo optimise** les instructions crées en résolvant les calculs dont on connais **toutes les variables**.

Évaluation à la compilation

`int i = 3 + 3` → `int i = 6`

`if (((3 + 6) == 9) && True)` → `if (True)`

Différentes options

Au lancement, Apollo donne plusieures options pour différentes fonctionalitées:

compiler et exécuter

compiler dans un exécutable

lancer un exécutable

donner la liste d’instructions de l’exécutable

...

Différentes options

run	→ compile et execute
build	→ compile dans un binaire
launch	→ execute un binaire
disassamble	→ intructions du binaire

Exemple de Code


```
@fibonacci(int a, int b, int stop) int {
    if stop == 0 {
        return (a + b);
    }
    return (@fibonacci(b, (a + b), (stop - 1)));
}

@main() int {
    int result = @fibonacci(1, 2, 3);
    print (result);
    return 0;
}
```

Tester le projet

Pour tester le langage apollo n’hesitez pas à accéder aux sources, compiler à l’aide de “make” puis de suivre les instructions pour executer les différents exemples dans le fichier “examples/run-examples.md” des sources.

Merci de votre lecture

arthur.aillet@epitech.eu
auguste.frater@epitech.eu
julian.scott@epitech.eu
ludovic.de-chavagnac@epitech.eu

EPITECH.
L'ECOLE DE L'EXPERTISE INFORMATIQUE