Typologie des langages
Résumé de Partiels
Fanny Riols

Langages :

Inventeurs :
Douglas Engelbart est l'inventeur de la souris.
Xerox (société) est à l'origine des systèmes de fenêtrage, de la souris et de l'imprimante laser.
Andrew Appel : Tiger.
John Backus (1924 - 2007) : FORTRAN, FP, APL.
Oke-Johan Dahl : Simula, Beta.
Alan Kay : Smalltalk.
Donald Knuth : TeX.
John McCarthy : Lisp.
Kristen Nigaard : Simula, Beta.
Dennis Ritchie : C.
Bjarne Stroustrup : C++.
Jean Ichbiah : Ada 83.
Larry Wall : Perl.
Niklaus Wirth : Algol W, Pascal, Euler, Modula-2, Oberon.
Grace Hopper : COBOL.

Typé :
C et OCaml sont typés *statiquement*.
JavaScript est typé *dynamiquement*.

Début :
BNF signifie *"Backus Naur Form"*.
Algol n'a pas été influencé par *Simula*.
IBM a fêté son siècle d'existence en 2011.
On doit if then else à *Algol*.
C#, Perl et Pascal UCSD repose sur une machine virtuelle pour assurer l'exécution du code, contrairement à *Go*.
L'*ALGOL* (*ALGOrithmic Language*) a introduit la notion d'instruction composée (compound statement) : *groupe d'instruction traité comme s'il s'agissait d'une seule*

instruction.
FORTRAN a montré au monde les bénéfices des langages de haut-niveau compilés.

Fonctions :

Fonctions récursives :

Le support des fonctions récursives nécessite *une pile*.

Portée :

La portée statique définit la visibilité d'une entité en fonction de sa *position dans le code source*.
La portée dynamique étend la *durée de vie* d'une entité : sa portée est *globale* et sa durée de vie *dynamique*.
La gestion des portées est la différence notable entre une table de symboles et un tableau associatif.
La portée d'un identificateur est la *zone de programme depuis laquelle il est accessible*.

Structure de blocs :

Les variables non locales sont permises grâce à la *structure de blocs*.
La structure de blocs est la *possibilité dans un langage de définir des blocs d'instructions*, eux-mêmes pouvant être imbriqués dans d'autres blocs. Algol (l'a introduit), Tiger, Lisp, Objective Caml et Haskell disposent de cette fonctionnalité.

Tiger :

En Tiger, nil est une *valeur qui peut avoir plusieurs types*.
Tiger supporte les *fonctions imbriquées*.

C et C++ :

En C, "int foo = getc() + getc() * getc()" dépend de la *plateforme et/ou de l'implémentation*.
En C et C++, on peut avoir des *pointeurs non initialisés*.
En C, les pointeurs sont dangereux car il s'agit d'une adresse mémoire qui ne correspond *pas forcément à un objet* : rien ne garantie qu'une valeur manipulée par pointeur soit valide.
En C++, les références sont des *pointeurs policés*, car une référence en C++ est également une adresse mémoire mais qui doit être initialisée via un objet existant. Une valeur manipulée par référence est valide. C'est donc plus sûr qu'un pointeur.
Les arguments formels d'une fonction sont normalement placés *sur la pile*.
En général, lorsqu'on définit un constructeur non trivial en C++, dans une classe qui possède des attributs de type pointeurs, on écrit systématiquement *un operator=*.

La liaison retardée en C++ est liée aux *méthodes polymorphes*.
La liaison dynamique en C++ a un rapport avec *"virtual"*.
La résolution des appels "virtual" nécessite la *connaissance du type des contenus*.
La résolution de la surcharge nécessite la *connaissance du type des contenants*.
Les "visiteurs" permettent d'*implémenter le "dispatching" une fois pour toute*.

Java :
En Java on peut avoir des références non initialisées.

Union :
En C, les unions sont dangereuses car une union définit des membres, mais la valeur d'au plus un membre est peut être stockée à un moment donné. Or le langage C n'offre aucun mécanisme pour connaître le membre réellement contenu dans l'union : ni à la compilation, ni à l'exécution. L'utilisateur doit donc préciser le membre qu'il veut utiliser : erreurs au typage faible !
En C++, une union peut contenir :

- des membres ayant un constructeur, un destructeur ou un opérator=, ou des tableaux de tels membres
- des attributs déclarés static
- des membres références
- des méthodes virtuelles
- → L'initialisation d'une union est assez différente de celle d'une classe ou d'une struct, et introduit ainsi bcp d'ambiguïté.

Mécanisme généralisant les unions de C pour le C++ : présent dans la biblio Boost via la classe paramétrée boost::variant<> → permet de définir des classes qui s'apparentent à des unions, où le type de chaque membre est un paramètre de boost::variant<>. La construction par défaut crée un variant contenant un membre du premier type. La classe fournit également des constructeurs pour chaque type de membres. L'affectation d'un variant à un autre variant effectue des affectations ou des constructions/destructions de membres, selon que les types des contenus dans les deux variants sont les mêmes ou non. La destruction d'un variant provoque la destruction de son contenu.
De telles fonctions devraient traiter chaque cas séparement :

```
type foo = Int of Int | Str of String.t::
let print_foo = function
        Int i → print_int i
        Str s → print_string s
;;
```

Enumérations :

Créer des types « ensembles » exhaustifs avec relativement peu de valeurs, et distincts des autres types (que ce soient des types énumérés ou pas). Des exemples classiques sont des énumérations représentant les jours de la semaine, les noms des mois, des noms de tokens, les booléens, etc. Elles sont très faiblement typées, puisque les étiquettes d'une enum sont implicitement convertibles en entiers. Au point que deux étiquettes de deux énumérations différentes peuvent être confondues! Pour information, le code suivant compile sans aucun warning, malgré un niveau de vérification élevé (option -Wall et -Wextra de g++).

Passage par nom :

Passage par nom = politique de passage d'argument dans laquelle la valeur symbolique de l'argument effectif est passée à la fonction appelée, au lieu de sa valeur.
En C++, on simule le passage par nom avec des macros.
Pour le code :

```
var     t       : integer
        foo     : array [1..2] of integer;

procedure     shoot_my (x : Mode integer);
begin
        foo[1] := 51;
        t       := 2;
        x       := x + 69;
end;

begin
        foo[1]  := 1;
        foo[2]  := 2;
        t       := 1;
        shoot_my (foo[t]);
end.
```

Mode (passage par…)	foo[1]	foo[2]	t	Explication
Valeur	51	2	2	Comme en C
Valeur-Résultat (Algol W)	51	70	2	Quand tu ressors de la fonction, tu remets le foo à la valeur de shoot_my, en ayant changer le x et le t

Valeur-Résultat (Ada)	70	2	2	Quand tu ressors de shoot_my, vu que tu passais avec x = foo[1] tu changes foo[1]
Référence	120	2	2	Comme on connait : x est sur l'adresse de foo[1]
Nom	51	71	2	x c'est "foo[t]". Donc quand on incrémente x, on incrémente foo[2]

Inlining :

Mise en ligne du corps d'une fonction (inlining) est une optimisation consistant à remplacer un appel de fonction ou de méthode par le corps de celle-ci au site d'appel.
En C++, une méthode ne peut être mise en ligne (inlined) si :

- Elle est récursive
- La définition de la fonction n'est pas visible depuis le site d'appel
- Elle est polymorphe
- Elle est appelée via un pointeur sur fonction
- Elle accepte un nombre variable d'arguments

Programmation fonctionnelle :

Langage fonctionnel :
Un langage est fonctionnel s'il permet de manipuler des fonctions comme n'importe quelle autre entité.
Un langage fonctionnel est dit pur lorsqu'il *proscrit tout effet de bord*.

Haskell :
L'évaluation paresseuse du langage Haskell permet d'exprimer des *"listes infinies"*.
cf langage Haskell.

Définition :
Une fermeture est une fonction qui *capture des références à des variables libres dans l'environnement lexical*.
Le patron de conception "Visitor" permet l'utilisation des *multiméthodes dans un langage objet qui en est démuni*.
Un design pattern est une *bonne solution à un problème connu*.

C++ :
En C++, on appelle objet/fonction un *objet disposant d'un operator()*.
Avec des objets fonctions, on peut implémenter des *"fonctions anonymes" dans un langage qui en est dépourvu*.

Java :
Java 7 ne propose *pas de fonctions anonymes*.

Programmation orientée objet :

Simula :
La notion de classe et d'objet a été introduit dans *Simula*.

Smalltalk :
En Smalltalk 76, on instancie/définit une (nouvelle) classe en envoyant un *message 'new' à la classe Class*.
Le typage en Smalltalk est *inexistant*.
En Smalltalk, une métaclasse est *une classe dont les instances sont des classes*.
En Smalltalk, les appels de méthodes ne sont *pas vérifiés statiquement*.

Eiffel :

En Eiffel, si une méthode prétend se substituer à une autre, il faut *accepter toutes les acceptations de son prédécesseur*. Donc les préconditions peuvent être *affaiblies* : *require else*.

En Eiffel, lorsqu'une méthode est redéfinie, les *postconditions peuvent être plus strictes* : *ensure then*.

Polymorphisme :

1. Un polymorphisme est la possibilité d'avoir plusieurs entités portant le même nom, mais ayant des attributs différents, sans entraîner aucune ambiguïté dans le comportement du programme.

STL s'appuie largement sur le polymorphisme paramétrique, lié à la généricité (en C++, lié à template).

Les langages OO introduisent le polymorphisme d'héritage ou d'inclusion (voire virtual en Simula et C++).

Les différents type de polymorphisme → générécité, abstraction, meilleur découplage interface/implémentation, extensibilité du code.

- Polymorphisme de coercition : capacité d'un type à se faire passer pour un autre : il s'agit d'une conversion de type implicite.
- Le polymorphisme ad hoc ou surcharge est la possibilité de donner le même nom à des routines différentes, qui se distinguent alors par leur signature.
- Le polymorphisme paramétrique permet l'écriture d'entités polymorphes vis-à-vis d'un paramètre statique.
- Le polymorphisme d'inclusion est celui induit par la relation « est un » entre objets (généralisation), et la faculté de redéfinir des méthodes poly- morphes (virtuelles en C++), dont la sélection est faite à l'exécution (liaison retardée, dynamic dispatch).
- Il n'y a pas de classe ni d'objet en C, donc pas de polymorphisme d'inclusion.

	Résolution statique	Résolution dynamique
concerne les types/classes	Paramétrique, Coercition	Inclusion
concerne les fonctions/méthodes	Paramétrique, Surcharge	Inclusion

Multiméthodes :

Multiméthodes = Sélection à l'exécution de la bonne fonction selon le type dynamique de n'importe quel argument, et non pas seulement l'argument (this). C'est un support à l'exécution de ce que la surcharge permet déjà de faire à la compilation.

Les multiméthodes permettent le polymorphisme dynamique sur plusieurs arguments de fonctions.

Les multiméthodes offrent une facilité d'implémentation : un premier gain, simple, c'est de ne plus être obligé "d'être dans la classe" pour pouvoir avoir la sélection dynamique. Ex : méthode print qui devient une fonction extérieure. Les visiteurs n'ont pas besoin des méthodes accept pour fonctionner. Les visiteurs simulent les multiméthodes. De plus, on a souvent besoin de sélection selon 2 arguments : par ex, l'arithmétique entre Numbers, où Numbers est une classe virtuelle, ou bien encore la recherche de l'intersection entre deux Shapes, où Shape est une classe abstraite.

CLOS (Common Lisp) et Perl 6 supportent les multiméthodes.

C++ :

En C++, le type dynamique d'un objet est un sous-type de son type statique.

Le type de retour n'entre pas en compte lors de la résolution d'une méthode surchargée, contrairement aux arguments de l'appel, au nom de la fonction et au qualitatif const de la méthode.

La surcharge est un mécanisme statique.

L'initialisation s'effectue à la construction d'un objet.

L'affectation est une opération qui a lieu après la construction d'un objet.

L'encapsulation est le fait de regrouper les données (attributs) et les algorithmes qui les manipulent (méthodes) au sein d'une même entité (classe).

Elle est réalisée grâce aux classes.

Le masquage de données est le fait d'empêcher l'accès aux données d'une classe (private, protected et public en C++).

Les méthodes virtuelles est un mécanisme dynamique.

La relation "est-un" s'exprime via l'héritage.

Programmation générique :

C++ :

Le C++ ne dispose pas de fonctionnalité conçue pour contraindre les paramètres des types paramétrés.

En C++, la spécialisation explicite permet de fournir une version alternative d'un template pour une combinaison de paramètres effectifs particulière.

En C++ 2011, auto ne peut pas être utilisé comme paramètre effectif d'un template.

En C++, const ne peut pas être utilisé comme paramètre effectif d'une classe paramétrée.
En C++, la fonction pair s'utilise : std::pair<T1, T2>.
En C++, on ne peut pas imposer des contraintes sur les paramètres d'un type.
En C++, un concept est un ensemble de règles sur un ou plusieurs paramètre(s).
Les templates de classes de C++ sont des générateurs de classes.
Les concepts du C++ ISO 2003 expriment des contraintes sur les paramètres de templates, et sont vérifiés implicitement par le compilateur.

Ada :
En Ada, on doit explicitement instancier les types paramétrés avant de pouvoir les utiliser.

Métaprogrammation :
La métaprogrammation statique permet d'exécuter des programmes à la compilation.