Variables

Vincent Archambault-Bouffard
IFT 2035 - Université de Montréal

Pour obtenir le code source de ce document

- https://github.com/archambaultv/IFT2035-UdeM
- vincent.archambault-bouffard@umontreal.ca

Plan du cours

- Comment faire référence aux données du programme ?
- Portée des variables

Comment faire référence aux données du programme ?

Référence aux données par leur adresse physique

Les données peuvent être :

- Dans la mémoire (RAM, cache, disque dur)
- Dans un registre

```
movl $10, -8(%rbp)
movl $15, -12(%rbp)
movl -8(%rbp), %ecx
addl -12(%rbp), %ecx
```

Code assembleur x86-64 pour 10 + 15

Référence aux données par leur adresse physique

✅ Très proche de la machine

❌ Peu lisible pour les humains

❌ Facile de commettre des erreurs

Référence aux données par une variable

Donner un nom aux données

```
x = 10
y = 15
z = x + y
```

Code Haskell pour 10 + 15

Référence aux données par une variable

Identificateur	Nom ou symbole
Variable	Espace de stockage associé avec un identificateur. L'espace de stockage peut être abstrait ou en lien avec un modèle de la mémoire.
Valeur	Donnée présente dans l'espace de stockage
Constante	Variable dont la valeur est fixe

Même identificateur ≠ même variable

```
x = let x = 5 in
      let x = 6 in
        x + x
```

```
x = let x1 = 5 in
      let x2 = 6 in
        x2 + x2
```

3 variables mais 1 identificateur (Haskell)

3 variables et 3 identificateurs (Haskell)

Variables et mémoire (virtuelle)

En C, l’espace de stockage est une mémoire virtuelle, sans registres ou caches

Les variables sont liées à des adresses mémoires

```
int x = 10;
int y = 15;
int z = x + y;
```

Code C pour 10 + 15

Variables et mémoire (virtuelle)

En C, l'espace de stockage est une mémoire virtuelle, sans registres ou caches

Les variables sont liées à des adresses mémoires

```
int x = 10;
int* addrX = &x;
int z = x + *addrX;
```

Code C pour 10 + 10 avec utilisation de pointeurs

Variables et valeurs

Il est possible de faire abstraction de la mémoire

L'espace de stockage est alors un concept abstrait

Il faut alors que le compilateur s'occupe de la gestion mémoire (garbage collector)

```
x = 10
y = 15
z = x + y
```

Code Haskell pour 10 + 15

Référence aux données par une variable

- ✅ Possible de garder une gestion manuelle de la mémoire (assez proche de la machine)
- ✅ Possible d’avoir une gestion automatique de la mémoire
- ✅ Lisible pour les humains
- ✅ Moins facile de commettre des erreurs

Portée des variables

Définition de portée

Portée Portion du programme où une variable est accessible par son identificateur

Portée lexicale (statique)

Portée délimitée textuellement

L'identificateur fait référence à la déclaration la plus proche dans le code source

```
foo x =
  let y = x + 1 in
  let x = y + 2 in
  x + y
```

Haskell utilise la portée lexicale

Portée lexicale (statique)

Un même identificateur peut être utilisé plusieurs fois dans des portées différentes

```
foo x =
  let y = 3 in
    y + 2

bar x =
  let y = 5 in
    y + 8
```

Réutilisation de l'identificateur x et y (Haskell)

Portée lexicale (statique)

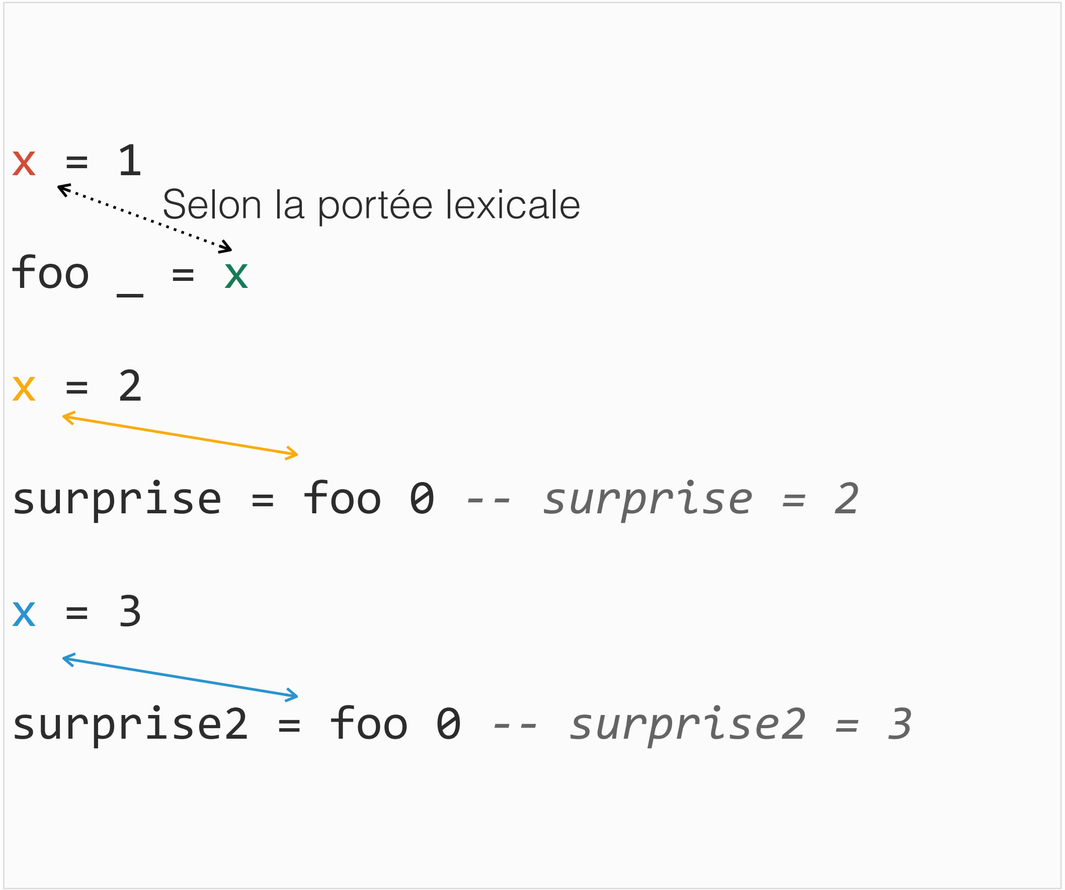
- ✅ Permet l’analyse statique du programme (automatique ou humaine)
- ✅ Sauf ombrage à une définition précédente, le choix de l’identificateur n’affecte pas le résultat

Portée dynamique

Portée délimitée par l'exécution du programme

L'identificateur fait référence à la déclaration active la plus récemment exécutée

Lorsqu'une fonction se termine ses déclarations locales ne sont plus actives



Pour chaque exécution de foo 0, la déclaration de x la plus récemment exécutée est utilisée (syntax Haskell)

Portée dynamique

Portée délimitée par l'exécution du programme

L'identificateur fait référence à la déclaration active la plus récemment exécutée

Lorsqu'une fonction se termine ses déclarations locales ne sont plus actives

```
x = 1

foo _ = x

bar _ = let x = 2 in
        x

surprise = let a = bar 0 in
           foo 0 -- surprise = 1
```

Bien que bar 0 implique une évaluation de la définition locale x=2, celle-ci n'est plus active lorsque foo 0 est exécuté

Portée dynamique

Une fonction a accès aux variables de l'environnement d'appel

```
foo _ = x

bar = let x = 3 in
      foo 0 -- foo 0 = 3

baz = let x = 2 in
      foo 0 -- foo 0 = 2
```

Pour chaque exécution de foo 0, la déclaration de x la plus récemment exécutée est utilisée (syntax Haskell)

Portée dynamique

Une fonction a accès aux variables de l'environnement d'appel

Permet le passage implicite d'argument

```
print text = write text destination

bar = let destination = stdout in
      print “Hello World”

baz = let destination = stderr in
      print “Erreur”
```

Passage implicite de l'argument destination (syntax Haskell)

Portée dynamique

- Un même usage peut faire référence à plusieurs définitions

  ✅ Passage implicite d’arguments qui changent rarement

  ❌ Facile de commettre des erreurs

❌ Le choix des identificateurs est important

❌ Ne permet pas une analyse statique du programme