Variables

Prof d'info

Lycée Thiers

Crédits

- Sur la notion de variable

Variables et constantes

- les *variables* sont des symboles qui associent un nom (l'identifiant) à une valeur. On peut les voir comme des tuples (id,val).

Variables et constantes

- les *variables* sont des symboles qui associent un nom (l'identifiant) à une valeur. On peut les voir comme des tuples (id,val).
- Dans les langages impératifs les variables peuvent changer de valeur au cours du temps (dynamique).

Variables et constantes

- les *variables* sont des symboles qui associent un nom (l'identifiant) à une valeur. On peut les voir comme des tuples (id,val).
- Dans les langages impératifs les variables peuvent changer de valeur au cours du temps (dynamique).
- Dans les langages fonctionnels, leur valeur est au contraire figée dans le temps (statique).

Variables et constantes

- les *variables* sont des symboles qui associent un nom (l'identifiant) à une valeur. On peut les voir comme des tuples (id,val).
- Dans les langages impératifs les variables peuvent changer de valeur au cours du temps (dynamique).
- Dans les langages fonctionnels, leur valeur est au contraire figée dans le temps (statique).
- Une *constante* est un identificateur associé à une valeur fixe. Cette valeur est choisie au début de l'*exécution* du programme et ne change pas par la suite.

Caractéristiques d'une Variable

Ce sont

nom

Caractéristiques d'une Variable

Ce sont

- nom
- valeur

Caractéristiques d'une Variable

Ce sont

- nom
- valeur
- type

Caractéristiques d'une Variable

Ce sont

- nom
- valeur
- type
- adresse

Caractéristiques d'une Variable

Ce sont

- nom
- valeur
- type
- adresse
- portée

Caractéristiques d'une Variable

Ce sont

- nom
- valeur
- type
- adresse
- portée
- visibilité

Caractéristiques d'une Variable

Ce sont

- nom
- valeur
- type
- adresse
- portée
- visibilité
- durée de vie

Caractéristiques d'une Variable

nom identifiant qui sert à désigner la variable dans la suite du programme

Caractéristiques d'une Variable

nom identifiant qui sert à désigner la variable dans la suite du programme

valeur suite de bits définissant le contenu de la variable

Caractéristiques d'une Variable

nom identifiant qui sert à désigner la variable dans la suite du programme

valeur suite de bits définissant le contenu de la variable

type façon dont est interprétée la suite de bits de la valeur.
Le type spécifie parfois aussi la longueur de la séquence : 8 bits, 32 ou 64...

Caractéristiques d'une Variable

nom identifiant qui sert à désigner la variable dans la suite du programme

valeur suite de bits définissant le contenu de la variable

type façon dont est interprétée la suite de bits de la valeur. Le type spécifie parfois aussi la longueur de la séquence : 8 bits, 32 ou 64...

adresse endroit de la mémoire où est stockée la variable.

Caractéristiques d'une Variable

portée notion qui concerne le code : partie du code source où la variable est accessible.

Caractéristiques d'une Variable

portée notion qui concerne le code : partie du code source où la variable est accessible.

visibilité qui peut utiliser la variable (*private*, *public*, *protected*), masquage de la variable par une autre.

Caractéristiques d'une Variable

portée notion qui concerne le code : partie du code source où la variable est accessible.

visibilité qui peut utiliser la variable (*private*, *public*, *protected*), masquage de la variable par une autre.

durée de vie notion qui concerne le temps d'exécution (*run time*) : quand la zone mémoire affectée a la variable pourra-t-elle être libérée sans risque ?

Portée vs Durée de vie

```
1  int a = 12;
2
3  int foo(){
4    int a = 13;
5    return a;}
6
7  int main(void){
8     printf ("a=%i\n",foo());
9     printf ("a=%i\n",a);
10   return 0;}
11
```

- La durée de vie du `a` ligne 1 est celle du programme lui-même.

```
$ gcc var.c -o var
$ ./var
a=13
a=12
```

Portée vs Durée de vie

```
1  int a = 12;
2
3  int foo(){
4    int a = 13;
5    return a;}
6
7  int main(void){
8     printf ("a=%i\n",foo());
9     printf ("a=%i\n",a);
10   return 0;}
11
```

- La durée de vie du `a` ligne 1 est celle du programme lui-même.
- Sa portée est l'ensemble des lignes sauf L4,L5

```
$ gcc var.c -o var
$ ./var
a=13
a=12
```

Typage dynamique vs statique

- Un typage est *dynamique* s'il est déterminé à l'exécution (ex : Smalltalk,Lisp,`Python`). Avantage : introspection possible (le programme PEUT s'examiner lui-même (PLUS de souplesse))

Typage dynamique vs statique

- Un typage est *dynamique* s'il est déterminé à l'exécution (ex : Smalltalk,Lisp,`Python`). Avantage : introspection possible (le programme PEUT s'examiner lui-même (PLUS de souplesse))
- Si le typage est déterminé à la compilation (explicitement par le programmeur ou automatiquement par inférence de types), alors il est *statique* (ex : C, OCAML).
  Avantage : génération de *bytecode* (c.a.d instructions machines) plus efficace en temps et mémoire.

Typage dynamique vs statique

En `Python`

Listing 1 – Python Console

```
>>> a=1
>>> type(a)
<class 'int'>
>>> a="toto"
>>> type(a)
<class 'str'>
```

Typage dynamique vs statique

En C

```c
int a =1;
a = 3.5;// on change le type implicitement

```

Après compilation :

```
$ gcc var4.c
var4.c:2:1: warning: data definition has no type or storage class
    2 | a = 3.5;
      | ^
var4.c:2:1: warning: type defaults to 'int' in declaration of 'a' [-Wimplicit-int]
var4.c:2:1: error: redefinition of 'a'
var4.c:1:5: note: previous definition of 'a' was here
    1 | int a =1;
      | ^
```

Typage dynamique vs statique

En C

```
1 int a =1;
2 char * a = "bonjour";// on change le type explicitement
3
```

Après compilation :

```
$ gcc -Wall var2.c
var2.c:2:8: error: conflicting types for 'a'
 char * a = "bonjour";
        ^
var2.c:1:5: note: previous definition of 'a' was here
 int a =1;
     ^
```

Typage fort vs faible

Un typage est *fort* lorsque le langage impose que les variables déclarées dans un type soient utilisées dans ce type. Il est alors plus difficile de faire avec une variable des choses pour lesquelles elle n'est pas prévue.

```
# let a= 1 in a + 0.1;;
Characters 16-19:
  let a= 1 in a + 0.1;;
                  ^^^
Error: This expression has type float but an expression
                was expected of type int
```

Il y a une distinction stricte entre les types (et donc on ne peut pas confondre des carottes et des navets, des **int** et des **float**...)
Exemple Ocaml

Typage fort vs faible

Si le typage n'est pas fort alors il est dit *faible* (langage C)

```c
int main(){
  int a = 1;
   printf("%f\n",a+0.1);
  return 0;}
```

Compilation puis exécution :

```
$ gcc -Wall var.c
$ ./a.out
1.100000
```

Pas de problème, `a` est utilisé comme un flottant alors que c'est un entier.

Notion de pointeur

- Les variables peuvent être de type entier, flottant, bouléen... mais on peut aussi décider que la valeur de la variable est une adresse de la mémoire. Une telle variable est alors appelée un *pointeur*.

Notion de pointeur

- Les variables peuvent être de type entier, flottant, bouléen... mais on peut aussi décider que la valeur de la variable est une adresse de la mémoire. Une telle variable est alors appelée un *pointeur*.
- Beaucoup de langages permettent la création dynamique d'adresse donc la manipulation de pointeurs (OCAML, Java, C, PHP..). Certains langages permettent de connaître cette adresse (C,C++), d'autres non (OCAML, Java, PHP).

Notion de pointeur

```c
int main(){
  int a = 1;// a est un entier
  int *pa = &a; //pa est un pointeur

   printf("valeur de a=%d\n",a);
   printf("adresse de a=%p\n",&a);
   printf("valeur de pa=%p\n",pa);
   printf("adresse de pa=%p\n",&pa);
   printf("valeur pointée par pa=%d\n",*pa);

  return 0;}
```

Après compilation puis exécution :

```
a=1
adresse de a=0x7fffd33da29c
valeur de pa=0x7fffd33da29c
adresse de pa=0x7fffd33da2a0
valeur pointée par pa=1
```

Nom des variables

- Les noms des variables obéissent à des règles syntaxiques qui dépendent des langages. En général, on ne court aucun risque à employer des lettres ou le underscore (touche 8).

Nom des variables

- Les noms des variables obéissent à des règles syntaxiques qui dépendent des langages. En général, on ne court aucun risque à employer des lettres ou le underscore (touche 8).
- Valides et invalides en C

```
1    int _var; float __var2; char * Var; int vArIAbLe, v_a_r;// valides
2    int +var; float 2a;// invalides
3    int a2; // valide
4
```

Nom des variables

- Les noms des variables obéissent à des règles syntaxiques qui dépendent des langages. En général, on ne court aucun risque à employer des lettres ou le underscore (touche 8).
- Valides et invalides en C

```
1    int _var; float __var2; char * Var; int vArIAbLe, v_a_r;// valides
2    int +var; float 2a;// invalides
3    int a2; // valide
4
```

- Le premier caractère ne peut être un chiffre, car cela permet de faciliter la compilation ou l'interprétation du programme en ôtant une ambiguïté :

Nom des variables

- Les noms des variables obéissent à des règles syntaxiques qui dépendent des langages. En général, on ne court aucun risque à employer des lettres ou le underscore (touche 8).
- Valides et invalides en C

```
1    int _var; float __var2; char * Var; int vArIAbLe, v_a_r;// valides
2    int +var; float 2a;// invalides
3    int a2; // valide
4
```

- Le premier caractère ne peut être un chiffre, car cela permet de faciliter la compilation ou l'interprétation du programme en ôtant une ambiguïté :
  - quand le compilateur lit un chiffre en 1ere position, il sait que les caractères qui suivront constitueront une valeur numérique.

Nom des variables

- Les noms des variables obéissent à des règles syntaxiques qui dépendent des langages. En général, on ne court aucun risque à employer des lettres ou le underscore (touche 8).
- Valides et invalides en C

```
1    int _var; float __var2; char * Var; int vArIAbLe, v_a_r;// valides
2    int +var; float 2a;// invalides
3    int a2; // valide
4
```

- Le premier caractère ne peut être un chiffre, car cela permet de faciliter la compilation ou l'interprétation du programme en ôtant une ambiguïté :
  - quand le compilateur lit un chiffre en 1ere position, il sait que les caractères qui suivront constitueront une valeur numérique.
  - De même, s'il lit une lettre ou un souligné, il saura qu'il a affaire à une variable.

Cycle de vie des variables

Cinq opérations courantes sur les variables

Déclaration Elle permet de déclarer un nom de variable, éventuellement de lui associer un type,

Suivant les langages, certaines de ces opérations sont fusionnées (déclaration et définition sont effectuées simultanément en C) d'autres sont impossibles (la suppression n'a pas vraiment de sens en C).

Cycle de vie des variables

Cinq opérations courantes sur les variables

Déclaration Elle permet de déclarer un nom de variable, éventuellement de lui associer un type,

Définition Elle permet d'associer une zone mémoire qui va être utilisée pour stocker la variable, comme lorsqu'on lui donne une valeur initiale,

Suivant les langages, certaines de ces opérations sont fusionnées (déclaration et définition sont effectuées simultanément en C) d'autres sont impossibles (la suppression n'a pas vraiment de sens en C).

Cycle de vie des variables

Cinq opérations courantes sur les variables

Déclaration Elle permet de déclarer un nom de variable, éventuellement de lui associer un type,

Définition Elle permet d'associer une zone mémoire qui va être utilisée pour stocker la variable, comme lorsqu'on lui donne une valeur initiale,

Affectation Elle consiste à attribuer une valeur à une variable,

Suivant les langages, certaines de ces opérations sont fusionnées (déclaration et définition sont effectuées simultanément en C) d'autres sont impossibles (la suppression n'a pas vraiment de sens en C).

Cycle de vie des variables

Cinq opérations courantes sur les variables

Déclaration Elle permet de déclarer un nom de variable, éventuellement de lui associer un type,

Définition Elle permet d'associer une zone mémoire qui va être utilisée pour stocker la variable, comme lorsqu'on lui donne une valeur initiale,

Affectation Elle consiste à attribuer une valeur à une variable,

Lecture consiste à utiliser la valeur d'une variable,

Suivant les langages, certaines de ces opérations sont fusionnées (déclaration et définition sont effectuées simultanément en C) d'autres sont impossibles (la suppression n'a pas vraiment de sens en C).

Cycle de vie des variables

Cinq opérations courantes sur les variables

Déclaration Elle permet de déclarer un nom de variable, éventuellement de lui associer un type,

Définition Elle permet d'associer une zone mémoire qui va être utilisée pour stocker la variable, comme lorsqu'on lui donne une valeur initiale,

Affectation Elle consiste à attribuer une valeur à une variable,

Lecture consiste à utiliser la valeur d'une variable,

Suppression Elle est réalisée soit automatiquement (Garbage Collector) soit par une instruction du langage (mot clé **del** en Python).

Suivant les langages, certaines de ces opérations sont fusionnées (déclaration et définition sont effectuées simultanément en C) d'autres sont impossibles (la suppression n'a pas vraiment de sens en C).

Cycle de vie des variables

Cinq opérations courantes sur les variables

Déclaration Elle permet de déclarer un nom de variable, éventuellement de lui associer un type,

Définition Elle permet d'associer une zone mémoire qui va être utilisée pour stocker la variable, comme lorsqu'on lui donne une valeur initiale,

Affectation Elle consiste à attribuer une valeur à une variable,

Lecture consiste à utiliser la valeur d'une variable,

Suppression Elle est réalisée soit automatiquement (Garbage Collector) soit par une instruction du langage (mot clé **del** en Python).

Suivant les langages, certaines de ces opérations sont fusionnées (déclaration et définition sont effectuées simultanément en C) d'autres sont impossibles (la suppression n'a pas vraiment de sens en C).

Initialisation

- *Initialisation* : fait d'allouer une valeur à une variable au moment de sa création.

Initialisation

- *Initialisation* : fait d'allouer une valeur à une variable au moment de sa création.
- L'initialisation est obligatoire ou non selon les langages. En C et Java, elle est facultative. En `Python` et Ocaml, obligatoire.

Initialisation

- *Initialisation* : fait d'allouer une valeur à une variable au moment de sa création.
- L'initialisation est obligatoire ou non selon les langages. En C et Java, elle est facultative. En `Python` et Ocaml, obligatoire.
- En C, une variable non initialisée a pour valeur le contenu de sa zone mémoire au moment de sa déclaration.

Initialisation

- *Initialisation* : fait d'allouer une valeur à une variable au moment de sa création.
- L'initialisation est obligatoire ou non selon les langages. En C et Java, elle est facultative. En `Python` et Ocaml, obligatoire.
- En C, une variable non initialisée a pour valeur le contenu de sa zone mémoire au moment de sa déclaration.
- en Java, une variable non initialisée se voit donner une valeur par défaut, qui sera toujours la même