Introduction à la Programmation 1

Séance 7 de cours/TD

Université Paris-Diderot

Objectifs:

- Représenter la mémoire sous la forme d'une table.
- Exécuter sur papier, un programme fait d'affectations, de séquences et de conditionnelles.

1 Expressions

La mémoire [Cours]

- Dans la mémoire sont stockées des valeurs.
- Une déclaration ajoute une variable à la mémoire avec sa valeur initiale.
- Une affection modifie la valeur d'une variable dans la mémoire.
- On représente la *mémoire* par une table associant chaque variable à sa valeur.
- Exemple :

x	y	z
2	7	42

  Dans cette mémoire, x vaut 2, y vaut 7, et z vaut 42.

Expressions de type `int` [Cours]

- Les valeurs de type `int` sont les entiers naturels compris entre -2^{31} et $2^{31}-1$. La valeur de 2^{31} est $2\,147\,483\,648$.
- Les opérations sur les expressions de type `int` sont : $+$, $-$, $*$, /,%.
- Il a des règles de priorité pour évaluer les expressions entières :
  - La multiplication, le quotient entier et le reste sont prioritaires sur l'addition et la soustraction.
  - Quand deux opérations de même priorité apparaissent dans une expression, c'est celle de gauche qui est effectuée en premier.
  - On peut mettre des parenthèses pour forcer une priorité.
- Par exemple :
  - 3 + 5 / 3 est évaluée comme 3 + (5 / 3)
  - 4 $*$ 1 / 4 est évaluée comme (4 $*$ 1) / 4
  - 2 / 3 $*$ 3 $-$ 2 est évaluée comme ((2 / 3) $*$ 3) $-$ 2
- Tant que le résultat d'une opération reste à l'intérieur des bornes, elle est évaluée comme en mathématiques.
- Au delà de ces bornes, on boucle :
  - $(2^{31}-1)+1$ s'évalue en -2^{31}
  - et $-2^{31}-1$ s'évalue en $2^{31}-1$
- Pour évaluer une expression qui contient des noms de variables, il faut aller chercher leurs valeurs respectives dans la table représentant la mémoire. On dit qu'on *évalue une expression par rapport à*

une mémoire car le résultat d'une expression dépend en général de la mémoire.

Exercice 1 (Exemples d'évaluation, ⋆)

1. *Quelles sont les valeurs des expressions entières suivantes :* 4 − 4 / 3, 3 ∗ 3 / 2, 4 − 3 − 2 *?*
2. *Étant donnée la mémoire suivante ;*

a	b	c
10	-5	2

comment sont évaluées les expressions qui suivent : a ∗ b − 10, a + c ∗ a, a + b − c, a + d *?*

□

Exercice 2 (Dépassement de bornes, ⋆⋆)

1. *Expliquez pourquoi l'évaluation de* $(2\,000\,000\,000+1\,000\,000\,000)/2$ *ne donne pas le résultat* $1\,500\,000\,000$.
2. *Écrire une fonction qui prend en argument deux entiers, et qui envoie comme résultat la moyenne entre les deux. Votre fonction doit aussi fonctionner dans le cas indiqué à la question (1).*

□

Expressions de type `String` [Cours]

— Pour rappel, l'unique opérateur dans les expressions de type String est l'opérateur de concaténation, noté +.
— De même façon que pour les entiers, si une expression contient une variable de type String qui est non déclarée alors l'évaluation échoue.

Exercice 3 (Encore des exemples d'évaluation, ⋆)
Étant donnée la mémoire suivante ;

ch	$x1$	$x2$	st
$"Hello"$	42	10	$"Bob"$

comment sont évaluées les expressions qui suivent : x1 ∗ x2 + 10, ch + st, st + " et Alice" *?* □

Expressions de type `boolean` [Cours]

— Les opérateurs utilisés dans ces expressions sont ||,&& et !.
— L'opérateur négation ! est prioritaire sur l'opérateur && (et) qui est lui même prioritaire sur l'opérateur || (ou). Ainsi, « b1 || b2 && b3 » est évalué comme « b1 || (b2 && b3) ».

Exercice 4 (Évaluation en présence de variables booléennes, ⋆)
Étant donnée la mémoire suivante :

x	y	$b1$	$b2$
10	8	$true$	$false$

comment sont évaluées les expressions qui suivent : (x > 0) && (y < 9), (x > 10) || (y == 8) && b1 *et* (b1 && b2) || (!b1 || !b2) *?* □

2 Exécution des déclarations, affectations et conditionnelles

Configurations [Cours]

— Un *compteur de programme* indique un numéro de la ligne du programme à exécuter. On le note souvent PC, de l'anglais *program counter*.
— Le couple formé d'une mémoire et d'un compteur de programme est une *configuration* - c'est ce que l'ordinateur, ou un humain, doit garder en "tête" pendant l'exécution d'un programme.
— Le compteur du programme indique le numéro de la ligne dont l'exécution a donné les valeurs des variables indiquées sur la même ligne.
— Exemple :

PC	x	y
12	2	7

Si l'exécution du programme se trouve dans cette configuration, ça veut dire qu'on vient d'exécuter la ligne 12, et qu'il y a maintenant deux variables en mémoire : x et y avec les valeurs 2 et 7.

Exécution de déclarations et d'affectations [Cours]

On peut *précisément* décrire l'exécution d'un programme à l'aide des configurations :

— On commence avec l'exécution de la première ligne du programme (dans le cas d'un programme complèt c'est la ligne suivant le `public static void main ...`).
— Puis une configuration *évolue* vers une autre en exécutant une ligne du programme.
— On s'arrête quand le compteur du programme pointe sur la ligne qui contient l'accolade fermante } de la fonction `main`.

Exemple d'exécution :

```
1 public class Simple {
2   public static void main (String[] arg) {
3     int x = 0;
4     int y = 10;
5     int z = 0;
6     x = 2 * y;
7     z = x + 1;
8   }
9 }
```

On a la séquence d'exécution suivante.

PC	x		
3	0		
PC	x	y	
4	0	10	
PC	x	y	z
5	0	10	0
PC	x	y	z
6	20	10	0
PC	x	y	z
7	20	10	21

Exercice 5 (État de la mémoire, ⋆)
Décrire l'état de la mémoire après les affectations suivantes :

```
1   public class Aff {
2     /* Fonction principale du programme */
3     public static void main (String[] args){
4       int x = 5;
5       int y = 6;
```

```
    }
  }
```

□

Précis d'exécution : déclarations et affectations [Cours]

- Étant donnée une configuration avec mémoire M et compteur de programme i, il faut définir comment trouver la configuration suivante.
- Après l'exécution d'une déclaration ou d'une affectation, le nouveau compteur de programme est $i+1$.
- La nouvelle mémoire dépend de ce que l'on trouve dans le programme à la ligne numéro i :
  - Déclaration avec initialisation « `int x = e;` » : Évaluer e par rapport à la mémoire M. Ajouter à la mémoire la variable x, avec comme valeur le résultat de e.
  - Affectation « `x = e;` » : Évaluer e par rapport à la mémoire M. Mettre le résultat comme nouvelle valeur de la variable x.
  - L'accolade fermante "}" de la fonction `main` : le programme s'arrête.

Exercice 6 (Première exécution, ⋆)
Décrire sur papier l'évolution de la mémoire lors de l'exécution d u programme suivant :

```java
public class ExoExec1 {

    public static void main(String[] args){
        int x1 = -77;
        String s = "Hey!";
        int y = 42;
        x1 = y/2;
        y = y+2;
        x1 = x1+y;
        s = s+s;
    }
}
```

code/ExoExec1.java

□

3 Exécution de programmes avec conditionnelles

Conditionnelles [Cours]

- Une conditionnelle est une instruction *composée*. On ne l'exécute pas d'un seul coup.
- Le premier pas consiste à exécuter la tête de la conditionnelle. Elle détermine laquelle des deux branches doit être exécutée par la suite. L'exécution de la tête d'une conditionnelle ne change pas la mémoire.
- Si l'exécution arrive sur la fin d'une branche alors on saute vers la ligne qui suit la conditionnelle.

Pour le programme ci-dessous :

```java
public class ExoPositif {
    public static void main (String [] arg) {
        int x = 2;
        if (x>1) {
            x = x+1;
            x = 2*x;
        } else {
            x = 42;
```

```
9             }
10            x=-x;
11        }
12    }
```

code/ExoPositif.java

On a la séquence d'exécutions suivante.

PC	x
3	2
PC	x
4	2
PC	x
5	3
PC	x
6	6
PC	x
7	6
PC	x
10	-6

Quant au programme ci-dessous :

```
1  public class ExoNegatif {
2      public static void main (String [] arg) {
3          int x = 2;
4          if (x==17) {
5              x = x+1;
6          } else {
7              x = x-1;
8          }
9          x = 2*x;
10     }
11 }
```

code/ExoNegatif.java

Il donnera la séquence d'exécution suivante.

PC	x
3	2
PC	x
4	2
PC	x
7	1
PC	x
8	1
PC	x
9	2

Précis d'exécution : instruction conditionnelle [Cours]

- Étant donnée une configuration avec mémoire M et compteur de programme i.
- Si on trouve à l'adresse i la tête d'une instruction conditionnelle « if (e) {» : On commence par évaluer la condition e par rapport à mémoire M.
  - Si le résultat est true alors le nouveau compteur de programme est $i+1$.
  - Si le résultat est false alors le nouveau compteur de programme est la ligne qui suit le « } else { » correspondant s'il existe, sinon la ligne qui suit le } correspondant.
- Si on trouve à l'adresse i une ligne « } else { » où "}" qui fait partie d'une conditionnelle alors le nouveau compteur de programme est le numéro de ligne qui suit la conditionnelle.

— Aucune de ces lignes ne modifie la mémoire.

Exercice 7 (Exécution conditionnelle, ⋆)
Décrire sur papier l'évolution de la mémoire lors de l'exécution du programme suivant :

```java
public class ExoExec2 {
    public static void main (String [] arg) {
        int x = 3;
        int y = 5;
        if (y>x) {
            x = y;
            y = y-1 ;
        } else {
            y = x;
            x = x+1;
        }
        x = x-y;
    }
}
```

code/ExoExec2.java

□

Exercice 8 (Exécution conditionnelle, ⋆⋆)
Décrire sur papier l'évolution de la mémoire lors de l'exécution du programme suivant :

```java
public class ExoExec3 {
    public static void main (String [] arg) {
        int x=1;
        if (x>0) {
            x = x+1 ;
            if (x==1) {
                x = 2*x;
            } else {
                x = 3*x;
            }
            x = 3*x;
        }
    }
}
```

code/ExoExec3.java

□

4 DIY

Exercice 9 (Deviner, ⋆⋆)

1. *Donner un programme qui pourra produire la séquence d'exécutions suivante :*

PC	a	
3	4	
PC	a	
4	16	
PC	a	b
5	16	2
PC	a	b
6	8	2
PC	a	b
7	8	-6

2. *Donnez un programme dont la suite d'instructions passera de l'état initial :*

PC	x
3	1

à l'état final :

PC	x	s	$s2$	$s3$	y
7	1	$"Bim"$	$"Pang"$	$"BimPang"$	3000

□

Exercice 10 (Analyser et corriger, ⋆)

Dans une agence d'assurance, le prix de base d'une assurance pour voiture est de 200 euros. Ce prix est modifié selon les critères suivants :

- *Si l'assuré a moins de 25 ans alors il faut rajouter 100 euros au prix.*
- *Ensuite si l'assuré a eu des accidents on rajoute au prix les montants suivants selon le nombre d'accidents :*
  - *pour un accident, 50 euros ;*
  - *pour deux accidents, 125 euros ;*
  - *pour trois accidents, 200 euros ;*
  - *pour quatre accidents ou plus, 100 euros + 50 euros par accident.*

1. *Combien devra alors payer un assuré de 23 ans, ayant eu deux accidents ?*
2. *On considère le programme suivant :*

```
1  public class ExoAssurance {
2      public static void main(String[] args){
3          int age = 23;
4          int accident = 2;
5          int prix = 200;
6          if ( age < 25 ) {
7              prix = prix + 100;
8          } else {
9              prix = prix + 0;
10         }
11         if ( accident <= 1 ){
12             prix = prix + 50;
13         }
14         if ( accident <= 2 ){
15             prix = prix + 125;
16         }
17         if (accident <= 3 ){
18             prix = prix + 200;
19         }
20         if (accident <= 4 ){
```

```
                prix = prix + 100 + accident * 50;
            }
        }
}
```

code/ExoAssurance.java

Donner l'évolution de la mémoire lors de l'exécution de ce programme. Quelle est la valeur de prix *à la fin de cette exécution ? Ce programme calcule-t-il le juste prix ?*

Écrire une fonction qui calcule le prix de l'assurance en fonction de l'âge et du nombre d'accidents. □

Exercice 11 (Écrire et analyser, ⋆)

1. *Écrire un programme Java qui à partir de la mémoire initiale suivante*

PC	$year$
3	1700

   teste si l'entier contenu dans la variable year *correspond à une année bissextile. Votre programme pourra mettre le résultat dans une variable booléenne* isLeap*. Pour rappel, une année est bissextile si elle est divisible par 4 mais pas divisible par 100, ou bien si elle est divisible par 400.*
2. *Donner l'exécution de votre programme en prenant comme valeur pour la variable* year *les entiers* 1600 *puis* 1900*.*

□

Exercice 12 (Erreurs, ⋆)

1. *Donner l'exécution du code suivant.*

```
        int a = 4;
        a = a*a;
        a = a*a;
        a = a*a;
```

code/ExoErreursA.java

   Refaire le même exercice avec au départ $a = 16$. Obtient-on le résulat attendu dans les deux cas ?
2. *Donnez l'exécution du code suivant.*

```
        int a = 4;
        int b = 2;
        a = a*a-b+2;
        b = b/(b-a);
```

code/ExoErreursB.java

   Donner des nouvelles valeurs pour a, b pour lesquelles ce code produira une anomalie lors de l'exécution. Donner l'exécution pas-à-pas pour ces valeurs.

□

‘

Exercice 13 (Racines, ⋆)

Les racines d'un polynôme quadratique $p(x) = ax^2 + bx + c$ sont les valeurs x tels que $p(x) = 0$. Les racines peuvent se calculer à l'aide de la méthode du discriminant. On pose $\Delta = b^2 - 4ac$.

Lorsque le discriminant est strictement positif ($\Delta > 0$), le polynôme admet deux racines $\begin{cases} x_1 = \frac{(-b+\sqrt{\Delta})}{2a} \\ x_2 = \frac{(-b-\sqrt{\Delta})}{2a} \end{cases}$.

Lorsque le discriminant est nul ($\Delta = 0$), il admet une unique racine $\frac{x=-b}{2a}$

Lorsque le discriminant est négatif ($\Delta < 0$), le polynôme n'a aucune racine réelle.

1. *Écrire un segment de code qui étant donnée les entiers a, b, c, produit une chaîne contenant un message décrivant les racines de ce polynôme. Ne calculez pas la racine carrée et ne cherchez pas à simplifier les expressions. Par exemple,*
   msg="La racine est $-3/8$", *ou* msg="Les deux racines sont (6+\u221A 4)/2 et (6−\u221A 4)/2" .
   Le code spécial \u221A *permet d'afficher le symbole* $\sqrt{}$.
2. *Trouver des valeurs pour a, b, c qui permettent de tester le bon fonctionnement de votre fonction dans tous les cas de figure.*
3. *Trouver des valeurs pour a, b, c qui provoqueront une anomalie à l'exécution. Exécutez le code pas à pas sur ces valeurs. Comment pouvez-vous changer votre programme pour éviter cette anomalie ?*

□