Chapitre 5 : Instructions

Informatique de base

2013-2014

Sup Galilée

Qu'est-ce qu'une instruction ?

Une instruction est un ordre donné à l'ordinateur de réaliser une suite d'actions dont chacune a pour effet de modifier le déroulement du programme, son environnement ou son état.

C'est à l'aide des instructions que peuvent être programmés les algorithmes. Par exemple, le calcul de n^p (où p est un nombre entier) peut être réalisé par une suite de p multiplications en exécutant les instructions d'un langage de programmation virtuel :

```
(1) lire n
(2) lire p
(3) affecter la valeur 1 à r
(4) affecter la valeur de 0 à i
(5) tant que i < p faire
(6)   début
(7)     affecter la valeur de r x n à r
(8)     affecter la valeur de i+1 à i
(9)   fin
(10) écrire la valeur de r
```

Qu'est-ce qu'une instruction ?

Les instructions offertes par C sont les suivantes :

- instruction vide,
- instruction "expression" (par exemple, les instructions 1, 2, 3, 4, 7, 8 et 10),
- bloc d'instructions (par exemple, l'instruction 6),
- instruction conditionnelle,
- instruction d'itération (par exemple, l'instruction 5),
- instruction de choix multiple,
- instruction d'interruption de séquence,
- retour d'une fonction.

Instruction vide

L'instruction ; est une instruction vide qui ne réalise aucune action.

Instruction “expression”

Si `exp` est une expression, alors `exp;` est une instruction qui réalise l'effet de bord de l'expression `exp`. Une telle instruction n'a donc de sens que si cet effet de bord n'est pas nul.

Par exemple, `12+4;` est une instruction qui ne déclenche aucune action. Elle est donc équivalente à l'instruction vide. Par contre l'instruction `x = 12 + 4;` en déclenche une : affecter 16 à x.

Bloc d'instructions

Si `decl1`, ..., `decl`m sont des déclarations de variables et `inst1`, ..., `inst`n sont des instructions, alors

```
{
decl1
...
declm
inst1
...
instn
}
```

est un bloc d'instructions. Un bloc d'instructions est donc composé d'une suite, éventuellement vide, de déclarations, suivie d'une suite d'instructions qui peuvent elles-mêmes être des blocs d'instructions.

Bloc d'instructions

Les variables déclarées dans un bloc sont des variables locales à ce bloc. Elles sont visibles dans ce bloc (à partir de leur déclaration) et dans tous les sous-blocs englobés qui ne les redéfinissent pas.

L'exécution d'un bloc d'instructions de déroule comme suit :

1. Les données déclarées dans le bloc sont empilées au sommet de l'environnement dans l'ordre de leur déclaration.

2. Les instructions sont exécutées séquentiellement jusqu'à la fin du bloc ou parce qu'une instruction `break` ou `return` est exécutée (provoquant la sortie du bloc).

3. À la sortie du bloc, les variables locales à ce bloc sont retirées de la pile. La sortie du bloc de la fonction `main` provoque l'arrêt du programme.

Bloc d'instructions : exemple

Soit le bloc d'instructions suivant :

```
{
int x = 5, d = 1;
float y;
  {
  float d = 0.5;
  y = x + d;
  }
}
```

Les déclarations de `x` et de `y` dans le bloc externe sont visibles dans le bloc interne. La déclaration de `d` dans le bloc interne masque celle du bloc externe : `d` est un entier dans le bloc externe et un flottant dans le bloc interne, et lorsqu'on se trouve en un point du bloc interne c'est à ce dernier qu'on se réfère en utilisant le nom `d`. Notons que la valeur initiale de `y` est indéterminée.

Bloc d'instructions : exemple

L'évolution de l'état du programme lors de l'exécution du bloc interne est la suivante :

Immédiatement avant l'exécution du bloc interne :

```
variable    y    _
variable    d    1
variable    x    5
```

Bloc d'instructions : exemple

L'évolution de l'état du programme lors de l'exécution du bloc interne est la suivante :

Immédiatement avant la sortie du bloc interne :

```
variable    d    0.5
variable    y    5.5
variable    d    (1)
variable    x    5
```

Bloc d'instructions : exemple

L'évolution de l'état du programme lors de l'exécution du bloc interne est la suivante :

Immédiatement après la sortie du bloc interne :

```
variable    y    5.5
variable    d    1
variable    x    5
```

Instruction conditionnelle

Une instruction conditionnelle permet de choisir l'instruction à exécuter parmi deux possibles, en fonction de la valeur d'une expression booléenne de choix.

Si `exp` est une expression, et si `inst`, `inst1` et `inst2` sont des instructions, alors

```
if (exp)
   inst
```

et

```
if (exp)
   inst1
else
   inst2
```

sont des instructions conditionnelles.

Instruction conditionnelle

La première réalise l'action suivante :
Si val(`exp`)$\neq 0$, **alors** exécuter l'instruction `inst`.

La seconde réalise l'action suivante :
Si val(`exp`)$\neq 0$, **alors** exécuter l'instruction `inst1`, **sinon** exécuter l'instruction `inst2`.

Par exemple :

```
if (mois >= 4 && mois < 10)
  saison = "chaude";
else
  saison = "froide";
```

Itération

Une instruction d'itération permet de répéter l'exécution d'une instruction tant qu'une condition est vérifiée comme par exemple ajouter 2 à la valeur d'une variable tant que cette valeur est inférieure à 100.

Trois instructions d'itération sont disponibles en langage C : `while`, `do...while`, et `for`.

Itération : `while`

Si `exp` est une expression et `inst` est une instruction, alors

```
while (exp)
  inst
```

est une instruction d'itération (ou boucle d'itération) dans laquelle `exp` est le test de continuation et `inst`, le corps de la boucle, est l'instruction à répéter.

L'action réalisée est la suivante : Le corps de la boucle (`inst`) est répété tant que le test de continuation est vrai (val(`exp`) $\neq 0$). En général le corps de la boucle est un bloc d'instructions.

Une instruction `while` doit être précédée d'une initialisation des variables dont dépend le test de continuation et dont les valeurs évolueront à chaque exécution du corps de la boucle.

Itération : `while`, exemple

Pour calculer la somme `s` des entiers de 1 à 9 peut être programmée comme suit :

```
{
int s, i;
s = 0;
i = 1;
while (i <= 9)
  {
  s = s+i;
  i = i+1;
  }
}
```

Itération : `do... while`

Si `inst` est une instruction et si `exp` est une expression, alors :

```
do
  inst
while (exp);
```

est une instruction d'itération. Comme dans le cas de l'instruction `while`, `inst` est le corps de la boucle et `exp` est le test de continuation.

L'action réalisée est la suivante : exécuter `inst` jusqu'à ce que val(`exp`)=0.

Remarquons que le corps de la boucle `inst` est exécutée au moins une fois.

Itération : `for`

Si `exp1`, `exp2`, `exp3` sont des expressions et si `inst` est une instruction, alors

```
for (exp1; exp2; exp3)
  inst
```

est une instruction d'itération équivalente à

```
exp1;
while (exp2)
  {
  inst
  exp3;
  }
```

Cette instruction intègre donc l'initialisation (`exp1`) et l'évolution (`exp2`) de la variable dont dépend le test de continuation (`exp3`).

Itération : `for`, exemple

Le calcul de la somme `s` des dix premiers entiers naturels peut se coder comme suit :

```
{
int s, n;
s=0;
for (n=1; n<=10; n=n+1)
  s=s+n;
}
```

Choix multiple

Une instruction de choix multiple permet de choisir une instruction à réaliser parmi un ensemble d'instructions possibles, en fonction d'une valeur d'une expression de choix.

Une instruction de choix multiple a la forme suivante :

```
switch (exp-choix)
  {
  inst-switch-1
  ...
  inst-switch-n
  }
```

où

- `exp-choix` est une expression.
- Le bloc entre accolades est le corps de l'instruction.
- Chaque inst-switch a l'une des trois formes suivantes :
  - `case exp-cas: inst` (`exp-cas` est une expression constante)
  - `default: inst` (`inst` est une instruction)
  - `inst`

Choix multiple

L'action réalisée par une instruction `switch` est la suivante :

Si dans le corps il existe une étiquette `case exp-cas:` telle que val(`exp-cas`)=val(`exp-choix`),
    alors se brancher à l'instruction qui suit immédiatement cette étiquette
sinon
  si dans le corps il existe une étiquette `default:`
  alors se brancher à l'instruction qui suit immédiatement cette étiquette
  sinon ne rien faire.
  fin-si
fin-Si

Une fois qu'un branchement a été effectué, les instructions sont exécutées en séquence jusqu'à la dernière instruction du corps. Pour éviter d'enchaîner l'exécution de deux cas exclusifs consécutifs, il faut terminer le premier par une instruction `break` qui a pour effet d'abandonner l'exécution de l'instruction.

Rupture de séquence

L'instruction `break;` provoque l'abandon de l'instruction en cours. Elle ne peut apparaître que dans le corps d'une instruction `while`, `do... while`, `for` ou `switch`.

Retour d'une fonction

Si `exp` est une instruction, alors `return exp;` est une instruction qui provoque, comme on le verra plus tard, l'abandon de l'exécution du bloc qui constitue le corps d'une fonction et retourne la valeur val(`exp`) de l'application de cette fonction.