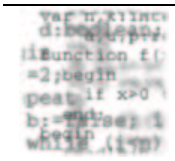

TurboPascal - Prépa HEC Ipecom

Tests - Boucles

notes de cours - vendredi 22 mars

1 Préalables

1.1 Conditions

De nombreuses structures de test ou boucle exécutent une instruction si une condition est vérifiée. Cette condition est vraie ou fausse, ce qui fait d'elle un booléen valant `TRUE` ou `FALSE`.

Les opérateurs simples servant à exprimer une condition sont les opérateurs arithmétiques `=`, `>`, `<`, `>=`, `<=` et `<>` [les trois derniers signifient $\geq$, $\leq$ et $\neq$]. Par exemple on peut écrire `B:=1>2;` et la variable booléenne `B` aura la valeur `FALSE`; si `a` et `b` sont des variables numériques, on peut écrire `B:=a=b` ou `B:=a<=(b+2)`.

Ces opérateurs peuvent se regrouper à l'aide des parenthèses `( )`, et se composer avec les opérateurs logiques `AND`, `OR` et `NOT`. La condition `(a=b)OR((a>c)AND(NOT(c>d)))` sera alors vérifiée si et seulement si `a` et `b` ont une valeur égale, ou `a` a une valeur strictement supérieure à `c` et `c` n'a pas une valeur strictement supérieure à `d`.

Remarques: `OR` n'est pas un OU exclusif, et il existe souvent plusieurs manières de formaliser une même condition.

1.2 Blocs d'instructions

On peut regrouper dans un programme TPascal plusieurs instructions en une seule, un bloc d'instructions; ces instructions seront toutes traitées à la suite, un peu comme une seule instruction se subdivisant en plusieurs. Cela n'est d'aucune utilité dans un programme simple, séquentiel [chacune des instructions du programme étant exécutée une fois, dans l'ordre], mais devient très utile dès la mise en place d'instructions complexes [tout le bloc d'instructions sera exécuté sous la même condition].

On écrit un bloc de la manière suivante, sans `;` après `begin`:

`begin` *instruction-1* `;` *instruction-2* `; ...  end;`

2 Tests

2.1 `if then else`

Un test contruit avec l'instruction `if then else` permet d'exécuter une instruction ou un bloc d'instruction si une condition est vérifiée, et éventuellement une utre unstruction ou un autre bloc si cette condition n'est pas vérifiéee.

La syntaxe générale, ouverte par `if` et fermée par `;`, est

`if` *condition* `then` *instruction-oui* `else` *instruction-non* `;`

Voici comment elle est traitée par TPascal:

1. La *condition* est évaluée.
2. Si elle est vraie [si sa valeur est `TRUE`], alors on exécute *instruction-oui*, puis on passe à l'étape suivante. Si elle est fausse, [si sa valeur est `FALSE`], alors on exécute *instruction-non*, puis on passe à l'étape suivante.
3. On passe à la suite [l'instruction suivante, après le `;`].

REMARQUES:

- Comme dans toute la suite, les instructions peuvent être des instructions simples, des tests, des boucles, ou des blocs d'instructions.
- Il ne faut pas mettre de `;` avant la fin, donc pas avant le `else`, par exemple[1].
- La partie `else` *instruction-non* est facultative, et la syntaxe peut se limiter à `if` *condition* `then` *instruction-oui* `;`. Dans ce cas, rien ne sera exécuté si la *condition* n'est pas vérifiée.

2.2 `case`

Alors que l'instruction `if then else` ne permet qu'un choix binaire [oui/non, vrai/faux], un test construit avec l'instruction `case` permet d'exécuter une instruction ou un bloc d'instruction pour chacune des valeurs que peut prendre une expression [chacune des valeurs possibles de la variable `a` ou de `a*2+b`, poar exemple].

La syntaxe générale, ouverte par `case` et fermée par `end;`, est

`case` *expression* `of` *valeur-1* `:` *instruction-1* `;` *valeur-2* `:` *instruction-2* `;` ... *valeur-n* `:` *instruction-n* `;` `else` *instructionbis-1; instructionbis-2; ...* `end;`

Voici comment elle est traitée par TPascal:

1. La valeur de l'*expression* est évaluée.
2. Si elle est égale à *valeur-1*, alors *instruction-1* est exécutée, et on passe à l'étape suivante, sinon on passe directement à l'étape suivante.
3. Si elle est égale à *valeur-2*, alors *instruction-2* est exécutée, et on passe à l'étape suivante, sinon on passe directement à l'étape suivante, ...
4. Si l'*expression* n'est égale à aucune des valeurs précédentes [*valeur-1*, *valeur-2*, ... *valeur-n*,], alors *instructionbis-1; instructionbis-2; ...* sont exécutées, et on passe à l'étape suivante, sinon on passe directement à l'étape suivante.
5. On passe à la suite [l'instruction suivante, après le `end;`].

REMARQUES:

[1] Cependant, des `;` après des instructions qui seraient incluses dans un bloc d'instructions ne posent pas de problème, car elle ne sont pas 'au même niveau' que le `if`, mais au niveau inférieur, celui des subdivisions du bloc.

- Les valeurs testées peuvent être des valeurs numériques, ou des variables dont on testera alors l'égalité avec l'*expression*, ou des intervalles, du type `1..3` ou `-5.5..6.5`.
- Chacune des valeurs est testée, même si un test précédent a été concluant. Plusieurs instructions pourront donc être exécutées, par exemple si on les associe aux valeurs `5..7` et `4..6`, et si l'*expression* vaut `5.5`.
- La partie `else` *instructionbis-1; instructionbis-2; ...* est facultative, et la syntaxe peut se limiter à `case` *expression* `of` *valeur-1* `:` *instruction-1*`;` *...* *valeur-n* `:` *instruction-n*`; end;`. Dans ce cas, rien ne sera exécuté si l'*expression* ne correspond à aucune des valeurs testées.

3 Boucles

3.1 `for do`

Une boucle construite avec l'instruction `for do` permet de répéter l'exécution d'une instruction ou d'un bloc d'instruction un certain nombre de fois.

La syntaxe générale, ouverte par `for` et fermée par `;`, est

`for` *compteur*`:=`*debut* `to/downto` *fin* `do` *instruction*`;`

Voici comment elle est traitée par TPascal, avec le mot-clé `to`:

1. La valeur de la variable *compteur* est initialisée à *debut*.
2. Si *compteur*`>`*fin* alors on passe directement à l'étape 5, sinon on débute une boucle en passant à l'étape suivante.
3. L'*instruction* est éxécutée.
4. Le *compteur* est incrémenté [*compteur*`:=`*compteur*`+1;`], et c'est la fin de la boucle: on retourne à l'étape 2.
5. On passe à la suite [l'instruction suivante, après le `;`].

REMARQUES:

- Avec le mot-clé `downto`, le *compteur* est décrémenté à chaque tour de boucle [*compteur*`:=`*compteur*`-1;`], et on teste avant chaque tour de boucle si *compteur*`<`*fin*.
- Attention à ne pas modifier accidentellement le compteur dans la boucle.
- Le compteur doit être une variable déclarée.

3.2 `while do`

Une boucle construite avec l'instruction `while do` permet de répéter l'exécution d'une instruction ou d'un bloc d'instruction TANT QUE une condition est vérifiée.

La syntaxe générale, ouverte par `while` et fermée par `;`, est

`while` *condition* `do` *instruction*`;`

Voici comment elle est traitée par TPascal:

1. La *condition* est évaluée.
2. Si elle est vraie [si sa valeur est `TRUE`], alors on débute une boucle en passant à l'étape 3. Si elle est fausse, [si sa valeur est `FALSE`], on passe à l'étape 4.
3. L'*instruction* est éxécutée, puis la boucle se termine et on retourne à l'étape 1.
4. On passe à la suite [l'instruction suivante, après le `;`].

REMARQUES:

- Il faut s'assurer, afin d'éviter une boucle sans fin, que la *condition* sera FAUSSE après un certain nombre de tours de boucle.
- La *condition*, qui est une condition d'exécution, est évaluée AVANT la boucle; il est donc possible que l'*instruction* ne soit jamais effectuée, si la *condition* est fausse dès le début.

3.3 `repeat until`

Une boucle construite avec l'instruction `repeat until` permet de répéter l'exécution d'une instruction ou d'un bloc d'instruction JUSQU'À CE QUE une condition est vérifiée.

La syntaxe générale, ouverte par `repeat` et fermée par `;`, est

`repeat` *instruction-1* `;` *instruction-2* `; ...` *instruction-n* `; until` *condition* `;`

Voici comment elle est traitée par TPascal:

1. Une boucle débute par l'éxécution de `titinstruction-1;` *instruction-2* `; ...` *instruction-n* `;`, puis se termine et on passe à l'étape suivante.
2. La *condition* est évaluée.
3. Si elle est vraie [si sa valeur est `TRUE`], alors on quitte les boucles en passant à l'étape 4. Si elle est fausse, [si sa valeur est `FALSE`], on débute une nouvelle boucle en passant à l'étape 1.
4. On passe à la suite [l'instruction suivante, après le `;`].

REMARQUES:

- Il faut s'assurer, afin d'éviter une boucle sans fin, que la *condition* sera VRAIE après un certain nombre de tours de boucle.
- La *condition*, qui est une condition d'arrêt, est évaluée APRÈS la boucle; l'*instruction* est donc toujours effectée au moins une fois.

Les structures sont redondantes : il est toujours possible, au prix de quelques lignes supplémetaires, de remplacer un `while do` ou un `repeat until` par un `for do` avec des tests `if then else`. Cependant, l'une des structures sera toujours plus naturelle, plus proche de l'objectif cherché, et donc plus simple à utiliser.