Synthèse de la syntaxe de base FreePascal

Laurent Vercouter - STPI / INSA de Rouen

1 Préambule

Cette fiche propose une très synthétique description de la syntaxe des instructions de base de FreePascal.

Éléments de notation formelle :
- les symboles <> encadrent un terme à remplacer par un identifiant (sans les <>)
- les symboles [ ] encadrent des termes dont l'écriture n'est pas obligatoire (sans les [ ])
- le symbole ... indique une possible répétition de la partie précédente

2 Structure d'un programme Pascal

La structure de base du code source d'un programme Pascal est la suivante :

```
program <nom_programme>;

[ declarations ]

begin
   [ instructions]
end.
```

3 Déclaration de variables

La déclaration de variables se fait dans un espace de déclaration avec le mot-clé `var`. Après ce mot-clé, on écrit l'identifiant de la variable à déclarer suivi de : puis du type de donnée souhaité avant de terminer par un ;

Si plusieurs variables du même type sont à déclarer, tous leurs identifiants peuvent être écrits avant les : en les séparant par une virgule.

Si des variables de type différents sont à déclarer, on peut ne pas répéter le mot-clé `var` et les écrire après le ; terminant la déclaration précédente.

Écriture générale :

```
var <id1> [, <id2>, ...] : <type1>;
    [<id3> [, <id4>, ...] : <type2>;]
    [...]

Exemple :

var nom           : String;
    taille, poids : Real;
    age           : Integer;
```

Parmi les types de données fréquemment utilisés on trouve : `Boolean`, `Integer`, `Real`, `Char`, `String`, `Array <dimensions> of <type1>`

La définition de nouveaux types de données se fait avec le mot-clé `Type` et la syntaxe suivante :

```
Type <nouveau_type> = <type_existant>;
```

4 Opérateurs

Voici quelques opérateurs classiques des types de données simples cités ci-dessus.

Boolean		
Symbole	Écriture	Type du résultat
not	not <Boolean>	Boolean
or	<Boolean> or <Boolean>	Boolean
and	<Boolean> and <Boolean>	Boolean
xor	<Boolean> xor <Boolean>	Boolean
=	<Boolean> = <Boolean>	Boolean
<>	<Boolean> <> <Boolean>	Boolean

Integer		
Symbole	Écriture	Type du résultat
+	<Integer> + <Integer>	Integer
-	<Integer> - <Integer>	Integer
*	<Integer> * <Integer>	Integer
/	<Integer> / <Integer>	Real
div	<Integer> div <Integer>	Integer
mod	<Integer> mod <Integer>	Integer
char	char(<Integer>)	Char
=	<Integer> = <Integer>	Boolean
<>	<Integer> <> <Integer>	Boolean
<	<Integer> < <Integer>	Boolean
>	<Integer> > <Integer>	Boolean
<=	<Integer> <= <Integer>	Boolean
>=	<Integer> >= <Integer>	Boolean

Real		
Symbole	Écriture	Type du résultat
+	<Real> + <Real>	Real
-	<Real> - <Real>	Real
*	<Real> * <Real>	Real
/	<Real> / <Real>	Real
=	<Real> = <Real>	Boolean
<>	<Real> <> <Real>	Boolean
<	<Real> < <Real>	Boolean
>	<Real> > <Real>	Boolean
<=	<Real> <= <Real>	Boolean
>=	<Real> >= <Real>	Boolean

Char		
Symbole	Écriture	Type du résultat
ord	ord(<Char>)	Integer
succ	succ(<Char>)	Char
pred	pred(<Char>)	Char
=	<Char> = <Char>	Boolean
<>	<Char> <> <Char>	Boolean
<	<Char> < <Char>	Boolean
>	<Char> > <Char>	Boolean
<=	<Char> <= <Char>	Boolean
>=	<Char> >= <Char>	Boolean

String		
Symbole	Écriture	Type du résultat
+	<String> + <String>	String
=	<String> = <String>	Boolean
<>	<String> <> <String>	Boolean
<	<String> < <String>	Boolean
>	<String> > <String>	Boolean
<=	<String> <= <String>	Boolean
>=	<String> >= <String>	Boolean

5 Instructions conditionnelles

Les deux instructions conditionnelles sont `if` et `case of` avec pour syntaxe :

```
if <condition> then
    <instructions>
[else
    <instructions>] ;
```

```
case <variable_choix> of
  <valeur1> [, <valeur2> ...] : <instructions>;
  <valeur3> [, <valeur4> ...] : <instructions>;
  ...
  else <instructions>;
end;
```

Avec :

- `<condition>` qui est une valeur booléenne
- `<variable_choix>` qui est une variable de type `Boolean`, `Integer`, `Char` ou un type énuméré
- `<valeur1>`, `<valeur2>`, ... qui sont des valeurs du type de `<variable_choix>`
- `<instructions>` qui est une seule instruction ou un ensemble d'instructions, séparées par un point-virgule, et encadré par les mots-clés `begin` et `end`

A noter : il ne fait pas placer de point-virgule après l'instruction suivant le `then` dans le cas où une clause `else` existe.

6 Instructions itératives

Les trois instructions itératives sont `for`, `while` et `repeat` avec pour syntaxe :

```
for <compteur> := <valeur_initiale> to <valeur_finale> do
   <instructions>;

while <condition> do
   <instructions>;

repeat
   <instructions>
until <condition>;
```

Avec :

- `<condition>` qui est une valeur booléenne
- `<compteur>` qui est une variable de type `Integer`, `Char` ou un type énuméré
- `<valeur_ initiale>` et `<valeur_finale>` qui sont des valeurs du type de `<compteur>`
- `<instructions>` qui est une seule instruction ou un ensemble d'instructions, séparées par un point-virgule, et encadré par les mots-clés `begin` et `end`

Une écriture alternative dite descendante d'une boucle `for` avec `<valeur_initiale>` $\geq$ `<valeur_finale>` et où le compteur décroît à chaque itération est :

```
for <compteur> := <valeur_initiale> downto <valeur_finale> do
   <instructions>;
```