Chapitre 0 : Migration du langage Pascal vers le langage C

Divers

Caractéristique	Pascal	C ++
sensible aux majuscules et minuscules	non	oui

commentaires	`{ commentaire }` Ou
	`(* Commentaire *)`
	`// Commentaire jusqu'à la fin de ligne`
		`/ * Commentaire * /` ou
		`// Commentaire jusqu'à la fin de ligne`

les constantes de type chaîne	`'Une chaîne constante'`	`"Une chaîne constante"`
les constantes de chaîne avec des côtes.	`'C''est belle journée!'`	`"J'ai dit \"Bonjour \" pour vous."`
chaîne avec retour à la ligne	`'Bonjour' # 13`	`"Bonjour \n"`

Types fondamentaux de variables

Taille et type du variable	Pascal	Visual C++
booléen	`Boolean`	`bool` (native type) ou `BOOL` (Win32 API)
entier signé sur 1 octet	`ShortInt`	`signed char`
entier non signé sur 1 octet	`Byte`	`char` *, `unsigned char`
entier signé sur 2 octets	`SmallInt`	`short`
entier non signé sur 2 octets	`Word`	`unsigned short`
entier signé sur 4 octets	`LongInt` ou `Integer`	`int` ou `long`
entier non signé sur 4 octets	Non disponible	`unsigned int` or `unsigned long` ou `DWORD` (Win32 API)
Virgule flottante sur 4 octets	`Single`	`float`
Virgule flottante sur 6 octets	`Real` **	Non disponible
Virgule flottante sur 8 octets	`Double`	`double` ou `long double`
Virgule flottante sur 10 octets	`Extended`	Non disponible

Opérateurs

Operateur	Object	C++
incrémentation	Non disponible	`i++, ++i`
décrémentation	Non disponible	`i--, --i`

NON logique	`not`	`!`
adresse	`@`	`&`
division à virgule flottante	`/`	`/`
division entière	`div`	`/`
modulo (le reste)	`mod`	`^`
égal à (test logique)	`=`	`==`
différent à	`<>`	`!=`
ET logique	`and`	`&&`
OU logique	`or`	`||`
instruction conditionnelle abrégée	Non disponible	`n > 7 ? b = 5 : b = 10;`
simple affectation	`:=`	`=`
Additionner, soustraire et multiplier, … etc.	Non disponible	`+=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=`

Instructions

Instruction	**Pascal**	**C++**

if ... else	`if 7 = x then`
	`  y := 5 // ";" illégale ici`
	`else`
	`  y := 7;`
		`if (7 == x)`
		`  y = 5;  // Faut avoir point-virgule`
		`else`
		`  y = 7;`

if ... else if	`if 7 = x then`
	`  y := 5  // pas de ";"`
	`else`
	`  if 9 = x then`
	`  y = 8  // pas de ";"`
	`  else`
	`  y = 11;`
		`if (7 == x)`
		`  y = 5;  // avec ";"`
		`else`
		`{`
		`  if (9 == x)`
		`  y = 8;  // avec ";"`
		`  else`
		`  y = 11;`
		`}`
		`/ * Notez que "if (7 == x)» est préférable à «if (x == 7)". Le compilateur signal l'erreur si vous mettre accidentellement "if (7 = x)», mais pas si vous mettre «si (x = 7)», ce qui entraînerait une difficulté de citer la position de l'erreur d'execution.* /`

for	`for i := 0 to 6 do`	`for (i = 0; i < 7; i++)`
while	`while n = 7 do`	`while (n == 7)`

do...while / repeat...until
(A noter que les conditions de test sont logiquement opposé.)
	`repeat`
	`...`
	`until n <> 7;`
		`do`
		`...`
		`while (n == 7);`

switch / case	`case n of`
	`  0: str := 'alpha';`
	`  1: str := 'beta';`
	`  2: str := 'gamma';`
	`else`
	`  str := 'invalid';`
	`end;  // case`
		`switch (n)`
		`{`
		`  case 0 : str = "alpha";`
		`  break;`
		`  case 1 : str = "beta";`
		`  break;`
		`  case 2 : str = "gamma";`
		`  break;`
		`  default: str = "invalid";`
		`}  // switch`

énumération	`type` `  colors = (cyan, magenta, yellow);`	`enum colors {cyan, magenta, yellow};`
tableaux	`arr = array [5..54] of integer;` `// on peut utiliser n'importe qu'elles bornes`	`int arr[50];` `// on commence par l'indice zéro`

pointers	`i: integer;`
	`pi: ^integer;  // déclare un pointeur`
	`pi^ = 6;`
		`int i = 5;`
		`int *pi;  // déclare un pointeur`
		`pi = &i;`
		`*pi = 6;  // *pi est identique à i`

Allocation dynamique du mémoire	`var`
	`  p : ^integer;`
	`begin`
	`  new(p);`
	`...`
	`dispose(p);`
		`int * pi = NULL;`
		`int * parri = NULL;`
		`// créer d'abord le pointeur`
		`pi = new int;  // allocation mémoire`
		`parri = new int[50];`
		`...`
		`delete pi;`
		`delete[] parri;`

structures / enregistrements	`type`
	`  s_t = record`
	`  amt: integer;`
	`  payee: string;`
	`  end;  // record`
	
	`var`
	`  s, s2: s_t;`
		`struct s_t`
		`{`
		`  int amt;`
		`  char payee[100];`
		`} s;  // déclaration de s`
		`  // est optionnelle ici`
		
		`s_t s2;`

fonction avec passage de paramètres par valeur (copiés)	`function somefunction(m: integer)`
	`: integer;`
	`begin`
	`  ...`
	`  result := 7;  // on utilise result`
	`  // ou somefunction := 7;`
	`end;`
		`int somefunction(int m)  // pas de ";"`
		`{`
		
		`  ...`
		`  return 7;`
		`}`

fonction avec passage de paramètres par valeur par adresse/référence (et des paramètres constants)	`procedure someprocedure`
	`(var m:integer, const n: integer);`
	`begin`
	`  ...`
	`end;`
		`void someprocedure(int &m, constint n)`
		`{`
		`  ...`
		`}`