Chapitre 4 CHAINES DE CARACTERES

A la fin de cette partie, vous serez capable de:

- Utiliser à bon escient les différents types de chaînes de caractères;
- Utiliser les différentes routines de traitement des chaînes de caractères;

4.1 Introduction

Depuis plusieurs années, les applications numériques ne sont plus le seul domaine de prédilection de l'informatique. L'utilisation de logiciels devient toujours plus conviviale et naturelle. Le traitement d'informations non-numériques s'avère donc nécessaire en programmation. Parmi les types de données non-numériques nous avons déjà examiné les types booléen, caractère et énuméré. Mais lorsqu'un programme traite des mots, des phrases, ou plus généralement du texte, il faut pouvoir disposer d'un nouveau type de données, d'un emploi souple et efficace. Le type chaîne de caractères (string) répond à ces critères. Il est absent des premières implémentations de Pascal, y compris celle de N. Wirth. Mais actuellement tous les langages Pascal, dont Delphi, autorisent ce type de données, et disposent de plusieurs procédures et fonctions standard qui lui sont associées.

4.2 Type chaîne de caractères (string)

Une chaîne de caractères est une suite de caractères. Une variable de ce type est déclarée par le mot **string** suivi de la longueur maximale de la chaîne de caractères, entre crochets. Lorsque le mot **string** apparaît seul, une longueur maximale n'est pas déterminée:

```
var nom     : string[25];
    prenom  : string[30];
    adresse : string[50];
    etudes  : string;
```

fig. 4.1

A l'aide de ces déclarations, les affectations qui suivent sont possibles:

```
nom := 'Dupont';
adresse := '36, chemin d''enfer';
prenom := '';
etudes := 'Lettres';
```

Le contenu d'une chaîne de caractères doit être placé entre apostrophes. Si une apostrophe doit figurer dans une chaîne de caractères, il faut la doubler. Il n'y a pas de restrictions concernant les caractères qui peuvent figurer dans une chaîne de caractères. En revanche, il faut veiller à ne pas dépasser le nombre de caractères maximum figurant dans la déclaration, faute de quoi l'excédent est tronqué.

Analysons les caractéristiques de la variable **nom** après l'affectation qui suit:

```
nom := 'Pam';
```

La **longueur physique** vaut 25 et correspond au nombre figurant entre crochets lors de la déclaration (voir figure 4.1).

La **longueur logique** est le nombre de caractères effectivement occupés dans la variable **nom**. Cette longueur vaut 3.

Les types chaîne de caractères et caractère ne sont pas entièrement compatibles. En particulier, l'affectation d'un caractère à une chaîne de caractères est possible, la chaîne aura alors une longueur de 1.

```
var adresse : string[10];
    lettre  : char;
...
lettre := 'G';
adresse := lettre;
...
```

En revanche, l'affectation d'une chaîne de caractères, même de longueur 0 ou 1, à une variable de type caractère n'est pas autorisée. L'affectation suivante, relative à la variable **lettre**, n'est donc pas admise:

```
var adresse : string[10];
    lettre  : char;
...
adresse := '4';       { initialisation }
lettre  := adresse;   { affectation incorrecte }
...
```

De plus, si une chaîne de caractères est affectée à une deuxième chaîne plus courte une troncation est effectuée.

```
var mot1 : string[20];
    mot2 : string[10];
...
mot1 := 'La prochaine fois';
mot2 := mot1;   { mot2 contient la chaîne 'La prochai' }
...
```

Enfin, des expressions booléennes peuvent être construites à l'aide de chaînes de caractères et d'opérateurs relationnels:

```
nom < 'Dupont'
prenom = 'Nestor'
adresse <> ''
```

Les comparaisons de chaînes de caractères se font d'après l'ordre lexicographique, en suivant le code ASCII des caractères. Avec cette convention, la chaîne de caractères 'Dupont' est inférieure à la chaîne de caractères 'dupont'. Il en est de même pour 'abc' et 'abcdef', ainsi que pour 'Z' et 'a' ! En effet, le code de 'Z' est 90, alors que celui de 'a' est 97.

4.3 Procédures et fonctions relatives aux chaînes de caractères

Dans cette section, nous allons examiner les procédures et fonctions standard les plus courantes concernant les chaînes de caractères (d'autres procédures et fonctions seront abordées plus loin). Elles sont données par ordre alphabétique.

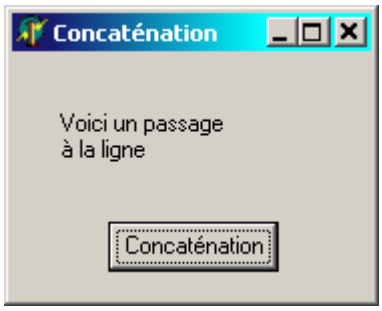
Concat **syntaxe: concat (st1, st2,..., stn)**

Cette fonction permet de concaténer deux ou plusieurs chaînes de caractères. Des constantes ou variables de type caractère peuvent également figurer dans la liste de paramètres de cette fonction. Le nombre de paramètres est variable, mais il faut éviter que la longueur de la chaîne

résultante ne dépasse la longueur déclarée pour la variable à laquelle elle est affectée. Les instructions qui suivent illustrent le fait de pouvoir concaténer indifféremment des caractères ou des chaînes de caractères:

```
procedure TForm1.Button1Click(Sender: TObject);
var st1      : string[30];
    resultat : string[60];
begin
  st1 := 'Voici un passage ';
  resultat := concat (st1, chr(13), 'à la ligne');
  label1.caption := resultat;
end;
```

Dans cet exemple un clic sur le bouton affiche le contenu de la variable **resultat** dans un Label:



Le TURBO Pascal dispose d'une autre forme de concaténation. Elle utilise l'opérateur de concaténation "+" et se présente de la manière suivante:

```
...
  st1 := 'Voici un passage ';
  resultat := st1 + chr(13) + 'à la ligne';
  label1.caption := resultat;
...
```

Ces instructions produisent le même résultat que celles de l'exemple précédent. A noter que chr(13) correspond au caractère "retour de chariot".

Copy **syntaxe: copy (st, position, nbre)**

Cette fonction retourne une chaîne de caractères extraite de la chaîne **st**; la position du premier caractère à extraire est fournie par **position**, alors que **nbre** indique le nombre de caractères à extraire. Les paramètres **position** et **nbre** doivent être de type entier.

```
st := 'Ceci est une chaîne très longue';
st2 := copy (st, 1, 19);
Edit1.text := st2;
```

L'exécution de ces instructions place la chaîne de caractères 'Ceci est une chaîne' dans Edit1.

Delete **syntaxe: delete (st, position, nbre)**

Cette procédure supprime un ou plusieurs caractères dans une chaîne de caractères, à partir d'une position donnée; **st** est l'identificateur de la variable contenant la chaîne de caractères, **position** est la position du premier caractère à supprimer et **nbre** est le nombre de caractères à supprimer. Les paramètres **position** et **nbre** doivent être de type entier.

```
st := 'Ceci est un petit exemple';
delete (st, 13, 6);
```

```
Edit1.text := st;
```

L'exécution de ces instructions place la chaîne de caractères 'Ceci est un exemple' dans Edit1.

Insert **syntaxe: insert (st1, st2, position)**

Cette procédure insère la chaîne de caractères **st1** dans la chaîne de caractères **st2** à partir de la position **position** (dans **st2**). Les paramètres **st1** et **st2** sont du type chaîne de caractères et **position** est de type entier.

```
St1 := 'de';
St2 := 'Delphi  Borland';
insert (st1, st2, 8);
Edit1.text := st2;
```

L'exécution de ces instructions place la chaîne de caractères 'Delphi de Borland' dans Edit1.

Length **syntaxe: length (st)**

Cette fonction fournit la longueur logique de la chaîne de caractères **st**, c'est-à-dire le nombre de caractères qu'elle contient.

```
st := 'Cette phrase contient 26 lettres';
longueur := length(st);
```

L'exécution de ces instructions place le nombre 32 dans la variable **longueur**.

Pos **syntaxe: pos (s_st, st)**

Cette fonction permet de déterminer si la chaîne de caractères **s_st** est contenue dans la chaîne **st**. Si c'est le cas, la fonction retourne la position où commence la chaîne **s_st**, à l'intérieur de la chaîne **st**. Si la chaîne **s_st** est absente de la chaîne **st**, le résultat est 0.

```
st := 'un deux trois quatre cinq';
pos1 := pos ('trois', st);
pos2 := pos ('six', st);
```

Après l'exécution de ces instructions **pos1** contient le nombre 9, alors que **pos2** contient le nombre 0.

Str **syntaxe: str (valeur, st)**

Cette procédure effectue la conversion de la valeur numérique contenue dans **valeur** en une chaîne de caractères **st**. Le paramètre **valeur** est de type entier ou réel et peut être suivi d'un paramètre d'écriture (dans notre exemple, on désire un nombre formaté sur 5 caractères au total, dont 1 décimale).

```
nombre := 456.34;
str (nombre:5:1, st);
Edit1.text := st;
```

L'exécution de ces instructions place la chaîne de caractères '456.3' dans Edit1.

Val **syntaxe: val (st, variable, code)**

Le but de cette procédure est de convertir la chaîne de caractères **st** en une variable numérique. Pour que cette conversion soit effective, le contenu de la chaîne de caractères doit correspondre aux règles d'écriture des nombres; de plus, aucun espace ne doit se trouver en première ou en dernière position. Le paramètre **variable** peut être de type entier ou réel. Après l'appel à cette procédure, si la conversion a pu être effectuée, la variable numérique **code** contient la valeur 0. Dans le cas contraire, la variable **code** contient la position du premier caractère de la chaîne **st** qui empêche la conversion, et le contenu de variable n'est pas défini.

```
st := '12.7';
val (st, nombre, code);
if code = 0 then
  Edit1.text := floattostr(nombre) + '  ' + inttostr(code)
else
  Edit1.text :=inttostr(code);
```

Le résultat de l'exécution de ces instructions place les nombres 12.7 et 0 dans Edit1. En revanche, si **st** avait été initialisé à '12a.7' l'exécution des instructions aurait placé le nombre 3 dans Edit1, correspondant à la position du caractère incorrect:

Remarque:

Il est possible d'accéder aux différents caractères d'une chaîne de caractères en indiquant, entre crochets, la position du caractère désiré:

```
...
var st : string[10];
    i  : integer;
...
st := 'remarque';
for i := 1 to length (st) do
  Label1.caption := Label1.caption + st[i] + chr(13);
...
```

Le résultat de l'exécution de ces instructions est l'affichage du mot "remarque" dans Label1, verticalement, un caractère par ligne.

4.4 Quelques précisions concernant les chaînes de caractères

Le type **string** est un type générique. En fait, Delphi gère trois types de chaînes de caractères:

Type	Longueur max.	Mémoire nécessaire	Utilisation
ShortString	255 caractères	2 à 256 octets	chaînes courtes, compatibilité ascendante
AnsiString	env. 2^{31} caractères	4 octets à 2 Goctets	caractères 8 bits (ANSI)
WideString	env 2^{30} caractères	4 octets à 2 Goctets	caractères Unicode

Dans les versions actuelles de Delphi l'indication du type **string** correspond, par défaut, au type **AnsiString**. Dans les cas où, pour des raisons de compatibilité avec d'anciennes versions de

Delphi ou de Pascal, on doit utiliser des chaînes courtes, il convient d'utiliser explicitement le type ShortString dans les déclaration, comme dans:

```
var nom : ShortString;
```

Dans la plupart des cas il est préférable d'utiliser le type générique **string** pour traiter des chaînes de caractères. Comme on peut le voir sur dans le tableau ci-dessus, une variable de type **string** (**AnsiString**) peut occuper, en mémoire, de 4 octets à 2 Goctets. Comme 2 Goctets représente une taille considérable, dans la pratique, la taille maximale est limitée par la quantité de mémoire disponible.

Le type **WideString** est analogue au type **string** (**AnsiString**) à la différence que chaque caractère de la chaîne est stocké sur deux octets (représentation Unicode) au lieu d'un. Pour une utilisation courante et un jeu de caractères standard (occidental), il est inutile d'utiliser le type WideString.

Remarques:

- L'apparition de nouveaux types de chaînes de caractères a été dictée par l'évolution des systèmes d'exploitation avec le support des caractères Unicode, ainsi que par le besoin de plus en plus fréquent d'utiliser de longues chaînes de caractères.
- Ces différents types peuvent sans autre être combinés dans les affectations et les expressions. Le compilateur prend en charge les conversions nécessaires de manière automatique et transparente.
- La gestion de la place mémoire occupée par les variables de type **string** s'effectue de manière dynamique. De ce fait la déclaration suivante:

  ```
  var nom : String;
  ```

  ne signifie pas que la place mémoire réservée à la variable **nom** est de 2 Goctets. En fait, la taille de la mémoire allouée à la variable **nom** varie en fonction du nombre de caractères contenus dans cette variable.
- Nous n'aborderons pas dans ce chapitre la manière dont sont stockées et représentées de manière interne les chaînes de caractères de type **string** (**AnsiString**).
- Quelques nuances, que nous n'aborderons pas ici, concernant la représentation interne ainsi que le traitement par le compilateur distinguent les types **AnsiString** et **WideString**.
- La VCL (Visual Component Library) de Delphi ne fait pas appel aux chaînes de type **WideString**.

Pchar et tableaux de caractères

Delphi supporte des chaînes de caractères appelées les chaînes à zéro terminal (AZT). Ces chaînes sont largement utilisées par les langages de programmation C et C++, et par Windows lui-même. Grâce au fait que Delphi supporte les chaînes à zéro terminal, ainsi que des fonctions de gestion de ces chaînes (dans l'unité SysUtils), il est facile d'interfacer un programme Delphi avec d'autres langages ou avec l'API Windows.

Une chaîne AZT est constituée d'une suite de caractères non **null**, suivis d'un caractère **null** (code ASCII #0). Ces chaînes n'ont pas d'indicateur de longueur séparé; le premier caractère NULL d'une chaîne AZT marque la fin de cette chaîne.

Une chaîne AZT se déclare de la manière suivante:

```
array[0..N] of Char
```

Un tableau de caractères à base zéro (dont le début commence à l'indice zéro) est compatible avec le type PChar. Cela signifie que partout où un PChar est attendu, il est possible d'utiliser à la place un tableau de caractères à base zéro.

Lorsqu'un tableau de caractères à base zéro est utilisé à la place d'une valeur PChar, le compilateur convertit le tableau de caractères en une constante pointeur dont la valeur correspond à l'adresse du premier élément du tableau.

Chaînes longues et chaînes à zéro terminal

La mémoire allouée pour une chaîne longue est terminée par un caractère **null** (qui ne fait pas partie de la chaîne, mais qui est stocké immédiatement après le dernier caractère de la chaîne). Grâce à cette marque de fin de chaîne, il est possible de transtyper une variable de type **AnsiString** en variable de type **Pchar** (pointeur vers une chaîne de caractères). Il suffit d'écrire PChar(S), où S est une variable **AnsiString**. Pchar(S) fournit un pointeur vers le premier caractère d'une chaîne longue.

Voici un exemple montrant cette possibilité. Si nous voulons afficher une boîte de message à l'aide des instructions suivantes:

```
var libelle : string;
begin
  libelle := 'Bonjour';
  messagebox (0, libelle, 'Message', mb_ok);
end;
```

une erreur se produit et le compilateur signale: "Types incompatibles 'String' et 'Pchar'". Il suffit alors d'effectuer la modification suivante pour que tout rentre dans l'ordre:

```
var libelle : string;
begin
  libelle := 'Bonjour';
  messagebox (0, Pchar(libelle), 'Message', mb_ok);
end;
```

Cet exemple montre en fait comment passer des chaînes longues à une fonction qui attend des paramètres chaîne à zéro terminal. Ce type de transtypage est très courant et même essentiel lorsque l'on doit appeler des fonctions de l'API de Windows qui, rappelons-le, sont entièrement écrites en langage C.

4.5 Routines de traitement des chaînes de caractères

Vous trouverez dans les tableaux qui suivent une description succincte des principales fonctions et procédures proposées par Delphi pour la gestion des chaînes de caractères. Toutes ces routines sont directement disponibles dans Delphi. Rappelons que l'on peut se procurer des librairies de routines en tout genre permettant de compléter l'assortiment proposé par Delphi.

Routines de traitement de chaînes de type string (chaînes Pascal)

Fonction	Description
AdjustLineBreaks	Transforme les ruptures de lignes dans une chaîne en séquences CR/LF.
AnsiCompareStr	Comparaison, en tenant compte des majuscules/minuscules, de deux chaînes.
AnsiCompareText	Comparaison, sans tenir compte des majuscules/minuscules, de deux chaînes.
AnsiLowerCase	Convertit des caractères en minuscules.
AnsiUpperCase	Convertit des caractères en majuscules.
CompareStr	Comparaison, en tenant compte des majuscules/minuscules, de deux chaînes.
CompareText	Comparaison, sans tenir compte des majuscules/minuscules, de deux

	chaînes.
Concat	Concatène une suite de chaînes.
Copy	Renvoie une sous-chaîne d'une chaîne.
Delete	Efface une sous-chaîne d'une chaîne.
DisposeStr	Libère une chaîne du tas.
FmtLoadStr	Charge une chaîne dans la ressource table de chaînes d'un programme.
Insert	Insère une sous-chaîne dans une chaîne.
IntToHex	Convertit un entier en hexadécimal.
IntToStr	Convertit un entier en chaîne.
IsValidIdent	Renvoie True si la chaîne spécifiée est un identificateur valide.
Length	Renvoie la longueur dynamique de la chaîne.
LoadStr	Charge la ressource chaîne depuis le fichier exécutable de l'application.
LowerCase	Met en minuscules la chaîne spécifiée.
NewStr	Alloue une nouvelle chaîne dans le tas.
Pos	Recherche une sous-chaîne dans une chaîne.
Str	Convertit une valeur numérique en chaîne.
StrToInt	Convertit une chaîne en entier.
StrToIntDef	Convertit une chaîne en entier ou à une valeur par défaut.
Trim	Supprime les espaces de début et de fin et les caractères de contrôle d'une chaîne donnée.
TrimLeft	Supprime les espaces de début et les caractères de contrôle d'une chaîne donnée
TrimRight	Supprime les espaces de fin et les caractères de contrôle d'une chaîne donnée.
UpperCase	Met en majuscules la chaîne spécifiée.
Val	Convertit une valeur chaîne en sa représentation numérique.

Routines de traitement de chaîne AZT

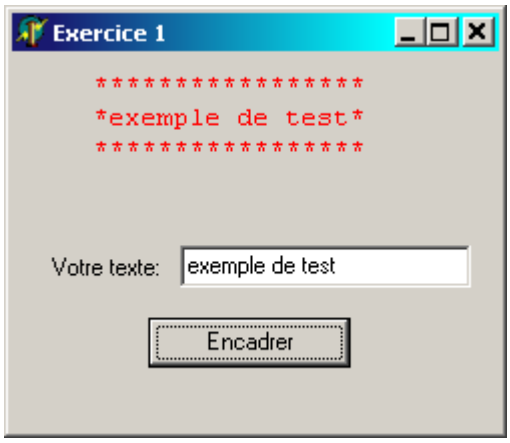
Fonction	Description
StrAlloc	Alloue une zone tampon d'une taille donnée sur le tas.
StrBufSize	Renvoie la taille d'une zone tampon de caractère allouée en utilisant StrAlloc ou StrNew.
StrCat	Concatène deux chaînes.
StrComp	Compare deux chaînes.
StrCopy	Copie une chaîne.
StrDispose	Dispose une zone tampon caractère allouée en utilisant StrAlloc ou StrNew.
StrECopy	Copie une chaîne et renvoie un pointeur à la fin de la chaîne.
StrEnd	Renvoie un pointeur à la fin d'une chaîne.

StrFmt	Formate une ou plusieurs valeurs dans une chaîne.
StrIComp	Compare deux chaînes sans tenir compte des majuscules/minuscules.
StrLCat	Concatène deux chaînes avec une longueur maximum donnée de la chaîne résultante.
StrLComp	Compare deux chaînes pour une longueur maximum donnée.
StrLCopy	Copie une chaîne jusqu'à une longueur maximum donnée.
StrLen	Renvoie la longueur d'une chaîne.
StrLFmt	Formate une ou plusieurs valeurs dans une chaîne avec une longueur maximum donnée.
StrLIComp	Compare deux chaînes pour une longueur maximum donnée sans tenir compte des majuscules/minuscules.
StrLower	Convertit une chaîne en minscules.
StrMove	Déplace un bloc de caractères d'une chaîne sur l'autre.
StrNew	Alloue une chaîne sur le tas.
StrPCopy	Copie une chaîne Pascal vers une chaîne à zéro terminal.
StrPLCopy	Copie une chaîne Pascal vers une chaîne AZT avec une longueur maximum donnée.
StrPos	Renvoie un pointeur sur la première occurrence d'une sous-chaîne donnée dans une chaîne.
StrRScan	Renvoie un pointeur sur la dernière occurrence d'un caractère donné dans une chaîne.
StrScan	Renvoie un pointeur sur la première occurrence d'un caractère donné dans une chaîne.
StrUpper	Convertit une chaîne en majuscules.

Exercice 4.1

Ecrire un programme dans lequel l'utilisateur peut taper un texte dans un Edit. Lorsque l'utilisateur clique sur le bouton "Encadrer" le programme doit afficher la chaîne de caractères dans un Label, encadrée par des astérisques. Il convient de prendre un Label assez haut pour contenir au moins trois lignes. Pour une bonne présentation du résultat, il convient également de choisir une police de caractères non proportionnelle pour le Label.

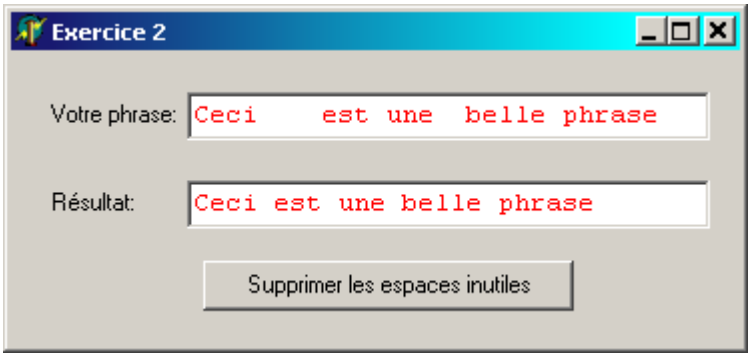
Voici l'aspect du programme:



Exercice 4.2

Ecrire un programme qui agit sur une phrase en remplaçant toutes les séquences de deux ou plusieurs espaces par un seul espace, et affiche la phrase obtenue. Afin de mieux visualiser les espaces, il est préférable de choisir une police de caractères non proportionnelle.

Voici l'aspect du programme:



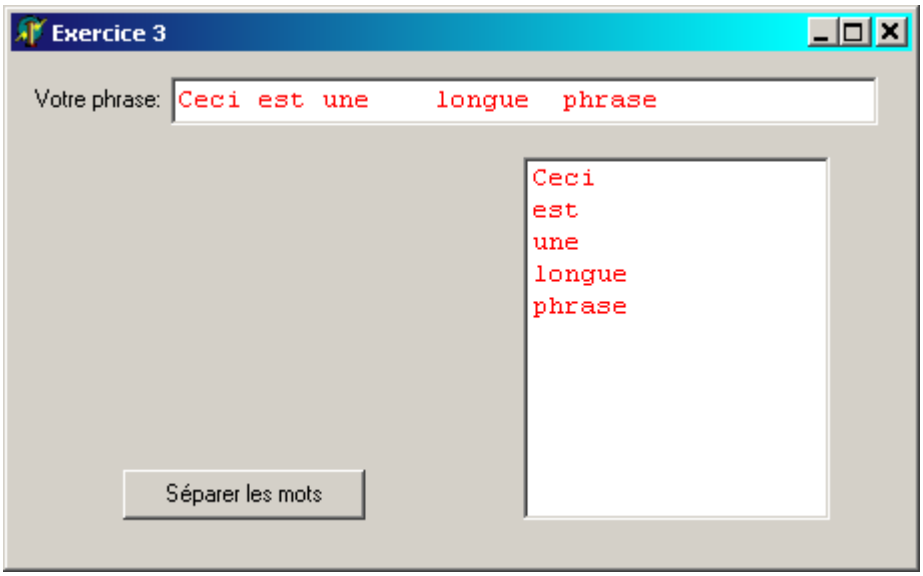
Exercice 4.3

Ecrire un programme qui affiche dans un Listbox les mots d'une phrase introduite par l'utilisateur. Chaque ligne du Listbox doit contenir un mot de la phrase. On considère que les mots sont séparés par un ou plusieurs espaces.

L'instruction qui permet d'ajouter une ligne dans un Listbox est:

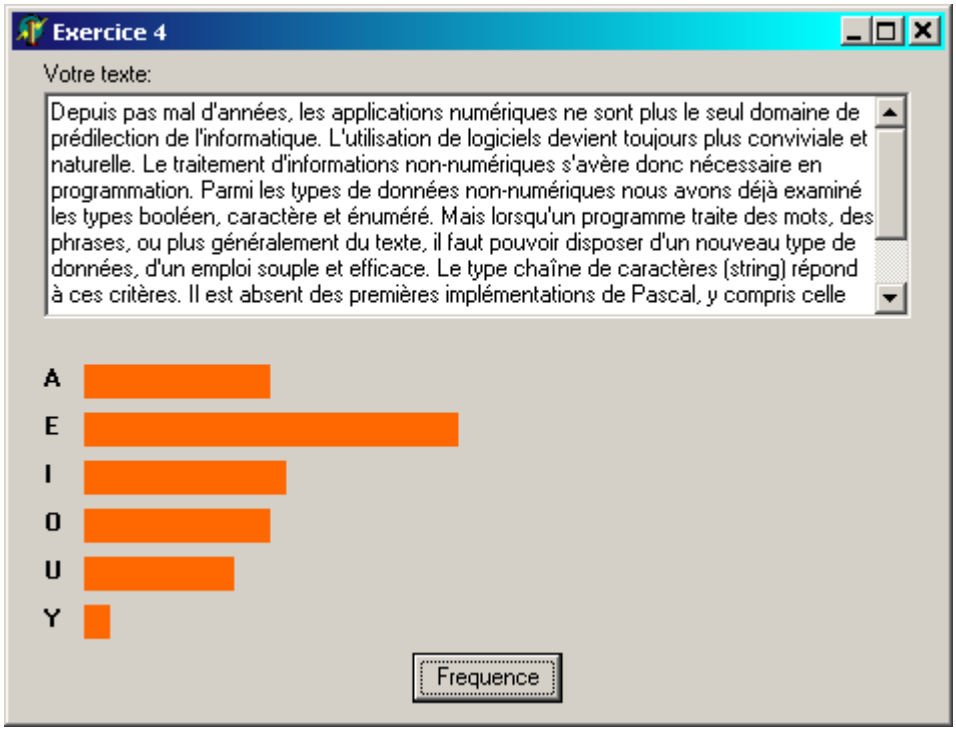
```
Listbox.Items.Add (mot);   // mot est la chaîne à ajouter
```

Voici l'aspect du programme:



Exercice 4.4

Ecrire un programme qui calcule la fréquence d'apparition des voyelles dans un texte que l'utilisateur fournit. Ce programme affiche ensuite un histogramme sous la forme suivante:



Le texte est placé dans un Memo (en fait dans Memo.text). L'histogramme est constitué de six Labels ayant une couleur se distinguant du fond gris de la fenêtre et dont la largeur varie en fonction de la fréquence d'apparition des voyelles.

Améliorer ce programme en ajoutant les fonctionnalités suivantes:

- afficher le nombre d'apparition de chaque voyelle
- effectuer des tests pour éviter que les barres de couleur ne sortent de la fenêtre
- au cas où une barre de couleur atteint le bord droit de la fenêtre, effectuer une renormalisation qui réduit proportionnellement la largeur de toutes les barres.