Cuerdas Borland C + + Builder

Los fundamentos de las cuerdas

Construcción y declaración de cadenas

Las operaciones de cadena en C ++ Builder se realizan principalmente utilizando una clase llamada AnsiString. La clase **AnsiString** no se deriva de **TObject** . Por lo tanto, tiene un alto nivel de independencia y flexibilidad de la aplicación o el control que quiere utilizarlo.

Es simplemente increíble el nivel de trabajo y el apoyo proporcionado por la VCL a sus cadenas y las operaciones relacionadas con el texto. Casi cualquier operación posible que se puede realizar en una cadena es compatible. Hay tantas funciones que vamos a revisar sólo las que se utilizan más en este libro, pero todas las funciones que se crearon en las bibliotecas son muy valiosas y puede ahorrar una enorme cantidad de escritura de código y dolor de cabeza.

Muchos controles utilizan propiedades AnsiString. Todos los controles que utilizan un título (formularios, paneles, etiquetas, etc) tienen su propiedad Caption definida como un valor AnsiString. Muchos otros controles como el cuadro de edición utilizan la clase AnsiString como la base de su texto. Basándose en estos dos hechos, ya ha utilizado e implementado los valores de AnsiString. En algunos otros escenarios tendrá que declarar y posiblemente inicializar una cadena antes de usarla.

Para declarar una cadena, utilice la palabra AnsiString seguida de un nombre C ++ válido. Aquí hay un ejemplo:

```
AnsiString País;
```

Puesto que AnsiString es una clase con su propio constructor, también puede declarar su variable con paréntesis vacíos, lo que llamaría al constructor de la clase. Aquí hay un ejemplo:

```
AnsiString City ();
```


Inicialización de cadenas

Hay dos formas principales de inicializar una variable AnsiString. Después de declararlo, puede asignar el valor deseado a la variable utilizando el operador de asignación. Aquí hay un ejemplo:

```
AnsiString País;
País = "Madagascar";
```

También puede inicializar la variable de cadena al declararla, de nuevo, utilizando el operador de asignación y proporcionando el valor deseado. Aquí hay un ejemplo:

```
Provincia de AnsiString ("Columbia Británica");
```

Una vez que haya definido la cadena puede utilizarla como mejor le parezca. Puede usarlo para cambiar el título de un control:

```
// ---------------------------------------------- ---------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString País;
        País = "Madagascar";
        Panel1-> Caption = País;
}
// ---------------------------------------------- ---------------------------
```

También puede usarlo para rellenar un control de edición:

```
// ---------------------------------------------- ---------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString Ciudad = "Antananrivo";
        Edit1-> Text = Ciudad;
}
// ---------------------------------------------- ---------------------------
```

Funciones de cadenas de propósito general

Vacío de cadenas

Funciones de propósito general son las que se realizan en cadenas, independientemente de cualquier otra consideración. Por ejemplo, antes de realizar cualquier operación en una cadena, a veces necesitará primero averiguar si la cadena contiene algo o está vacía. Eventualmente, usted decidirá qué hacer si la cadena está vacía.

Hay dos formas principales de comprobar si el contenido de un control basado en texto está vacío. Sólo puede utilizar el operador AnsiString "==" sobrecargado. Aquí hay un ejemplo:

```
// ---------------------------------------------- ---------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString Original = Editar1-> Texto;

        Si (original == "")
                Edit2-> Text = "¿Por qué Edit1 está vacío?";
```

```
}
// ---------------------------------------------- --------------------------
```

La clase **AnsiString** proporciona su propia función utilizada para comprobar si una cadena está vacía. Su sintaxis es:

```
Bool __fastcall IsEmpty () const;
```

Esta función miembro se puede utilizar para comprobar si un control basado en texto contiene nada pero no puede vaciar un control de texto. En el ejemplo siguiente se muestran dos formas de utilizar el **método AnsiString :: IsEmpty ()** :

```
// ---------------------------------------------- --------------------------
Void __fastcall TOKBottomDlg :: OKBtnClick (TObject * Sender)
{
        String UserName = edtUserName-> Text;

        If (Edit1.IsEmpty ())
                Panel1-> Caption = "Proporcione un nombre de usuario";
        If (Edit2-> Text == "")
                Panel1-> Caption = "Necesita escribir una contraseña";
        If (Edit3-> Text.IsEmpty ())
                Panel1-> Caption = "Tu cuenta no está completa";

        EdtUserName-> SetFocus ();
}
// ---------------------------------------------- --------------------------
```

La longitud de una cadena

Cuando una cadena ha sido inicializada o al menos no está vacía, tiene una longitud. Hay varias maneras de obtener o controlar la longitud de una cadena.

Para encontrar la longitud de una cadena, si la cadena es de la clase C string, primero puede convertirla usando la función **AnsiString :: c_str ()** , luego use la función strlen () para obtener su longitud:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Char * S = Edit1-> Text.c_str ();
        Edit2-> Text = strlen (S);
}
// ---------------------------------------------- --------------------------
```

Para obtener la longitud de una variable AnsiString, utilice el **método AnsiString :: Length ()** . Su sintaxis es:

```
Int __fastcall Longitud () const;
```

Esta función devuelve la longitud de la cadena como un entero. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString S = Editar1-> Texto;
        Edit2-> Text = S.Length ();
}
// ---------------------------------------------- --------------------------
```

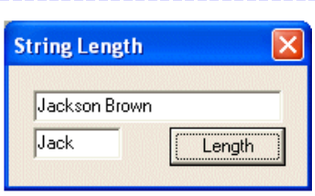
La clase AnsiString le permite imponer el número de caracteres de una cadena. Utiliza el siguiente método:

```
AnsiString & __fastcall SetLength (int NewLength );
```

Este método toma un argumento que es la longitud que desea que tenga la variable AnsiString. Si el argumento *NewLength* es menor que la longitud de la cadena, la cadena resultante sería los primeros caracteres *NewLength* de la cadena original. Si el valor *NewLength* es menor que la longitud de la cadena, el original se conservará y se asignará a la cadena resultante:

```
// ---------------------------------------------- ---
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
    AnsiString S = Editar1-> Texto;
    Edit2-> Text = S.SetLength (7);
}
// ---------------------------------------------- ---
```


```
// ---------------------------------------------- ---
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
    AnsiString S = Editar1-> Texto;
    Edit2-> Text = S.SetLength (4);
}
// ---------------------------------------------- ---
```



Trimming de cuerdas

Cortar una cadena es una operación que elimina los espacios iniciales o finales de una cadena. Para quitar cualquier espacio (vacío) en el lado izquierdo de una cadena, puede utilizar el **método AnsiString :: TrimLeft ()** . Su sintaxis es:

```
AnsiString __fastcall TrimLeft () const;
```

Si la cadena original tiene espacio a la izquierda, esta función lo quitará y devolverá una cadena que sea como la original sin el espacio principal. Si el original no tiene ningún espacio inicial, la función devolverá la misma cadena:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Edit2-> Text = Edit1-> Text.TrimLeft ();
}
// ---------------------------------------------- --------------------------
```

Otra función utilizada para realizar la misma operación es TrimLeft (). Su sintaxis es:

```
AnsiString _fastcall TrimLeft (const AnsiString S);
```

A diferencia del **método AnsiString :: TrimLeft ()** , la función (global) **TrimLeft ()** toma un argumento que es la cadena que necesita ser recortada. La función devuelve una nueva cadena que es la misma que la original omitiendo el espacio principal (si existe):

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Edit2-> Text = TrimLeft (Edit1-> Text);
}
// ---------------------------------------------- --------------------------
```

Para quitar cualquier espacio en el lado derecho de una cadena, puede utilizar el **método AnsiString :: TrimRight ()** . Su sintaxis es:

```
AnsiString __fastcall TrimRight () const;
```

Si la cadena original tiene espacio a su derecha, esta función lo quitará y devolverá la misma cadena sin ningún espacio al final. De lo contrario, se devolverá la cadena original:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
Edit2-> Text = Edit1-> Text.TrimRight ();
}
// ---------------------------------------------- --------------------------
```

Otra función utilizada para realizar la misma operación es **TrimRight ()** . Su sintaxis es:

```
AnsiString _fastcall TrimRight (const AnsiString S);
```

La función (global) **TrimRight ()** requiere un argumento como la cadena que debe recortarse. La función devuelve la cadena original sin el espacio final:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
Edit2-> Text = TrimRight (Editar1-> Texto);
}
// ---------------------------------------------- --------------------------
```

Otras funciones permiten combinar las dos últimas operaciones en una sola. Puede utilizar el **método AnsiString :: Trim ()** para quitar espacios en ambos lados de una cadena. Su sintaxis es:

```
AnsiString __fastcall Trim () const;
```

He aquí un ejemplo de cómo usar este método:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Edit2-> Text = Edit1-> Text.Trim ();
}
// ---------------------------------------------- --------------------------
```

Como alternativa, puede utilizar la función global Trim () para realizar la misma operación. Su sintaxis es:

```
AnsiString _fastcall Trim (const AnsiString S);
```

Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Edit2-> Text = Trim (Editar1-> Texto);
}
// ---------------------------------------------- --------------------------
```

Conversiones de cadenas

Tipos de datos C / C ++ Conversión a AnsiString

Un valor que el usuario escribe en un control como un cuadro de edición se considera una cadena. Esto se debe a que el compilador no puede asumir el tipo de valor que proporcionaría el usuario o el

cliente de un control de edición. Por esta razón, después de haber proporcionado un valor a un control que utiliza AnsiString como base de su contenido, si desea realizar cualquier operación matemática en la cadena debe convertir la cadena a un tipo de datos válido.

La clase AnsiString proporciona una gran cantidad de constructores que le permiten crear una cadena de cualquier tipo. Por ejemplo, puedes usarlo para declarar:

- Un carácter:
  AnsiString Símbolo = 'H';
- Un entero:
  AnsiString Int = 120;
- Un entero largo:
  AnsiString Longer = -73495745;
- Un valor de coma flotante:
  AnsiString WeeklyEarnings = 675.15;
- Un número de doble precisión:
  AnsiString WeeklyEarnings = 675.15;
  AnsiString Silver = 2.15e28;
- Una cadena:
  AnsiString GirlFriend = "Micheline Phoon";

Cualquiera de estas variables puede ser declarada usando sus constructores equivalentes: AnsiString Symbol ('H');

```
AnsiString Int (120);
AnsiString GirlFriend ("Micheline Phoon");
AnsiString WeeklyEarnings (675.15);
AnsiString más largo (-73495745);
AnsiString Silver (2.15e28);
```

Basado en las configuraciones de los constructores AnsiString, puede convertir cualquier valor y ponerlo a disposición de un control que utiliza una propiedad AnsiString. Por ejemplo, puede convertir y mostrar:

- Un carácter:
  char Sign = 'q';
  Edit1-> Text = AnsiString (Signo);
- Un interger:
  Número entero = 808;
  Leyenda-> Texto = AnsiString (Número);
- Un entero largo:
  largo Valor = 497783L;
  Panel1-> Caption = AnsiString (Valor);
- Un valor de punto **flotante** :
  Distancia de flotación = 1205,62;
  Label1-> Caption = AnsiString (Distancia);
- Un número de doble precisión:
  Double YearlyIncome = 24588;
  Edit1-> Text = AnsiString (YearlyIncome);
- Una cadena:
  AnsiString Food = "Mantequilla de Cacahuete";
  Button2-> Caption = AnsiString (Alimentación);

AnsiString y C-Strings

La clase AnsiString está configurada para reconocer cadenas con terminación nula de las funciones de cadena C clásicas. La clase AnsiString tiene un constructor que puede convertir una cadena C terminada en NULL en AnsiString. Gracias a este constructor, el AnsiString (const char * Source), puede declarar una cadena C y usarla como mejor le parezca:

```
// ---------------------------------------------- ---------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Char Status [] = "Estado del Empleado";
        Caption = Estado;
        Char * FullName = "Antoine Williams";
        Edit1-> Text = FullName;
}
// ---------------------------------------------- ---------------------------
```

Basado en este constructor, puede declarar una cadena C, manipular su valor y utilizarlo en su aplicación. Para convertir un valor AnsiString en una cadena terminada en null, utilice el **método AnsiString :: c_str ()** . La sintaxis de esta función es:

```
Char * __fastcall c_str () const;
```

Esta función es muy valiosa porque puede ayudarle a realizar varios tipos de operaciones de cadena.

Cajas de Cuerdas: Conversión de minúsculas a mayúsculas

Hay varias técnicas que puede utilizar para convertir una cadena de minúsculas a mayúsculas y viceversa. Se reconoce un carácter alfabético en minúsculas si es uno de los siguientes caracteres: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, Q, r, s, t, u, v, w, x, y, z. Por otro lado, un carácter califica como mayúscula si es uno de A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R , S, T, U, V, W, X, Y, Z. Todos los demás símbolos se ignoran incluso si en el teclado presionar Shift para escribirlos.

Para convertir un carácter en minúsculas o una cadena en mayúscula, puede utilizar la función de miembro **AnsiString :: UpperCase ()** . Su sintaxis es:

```
AnsiString __fastcall UpperCase () const;
```

Esta función miembro considera una variable AnsiString y examina cada uno de sus caracteres. Si un carácter es un carácter alfabético en minúsculas, se convertiría en mayúsculas. Si el carácter es un carácter alfabético en mayúsculas o no es un carácter alfabético, se mantendría "tal cual". Este método también considera la configuración regional del equipo que se utiliza, como se establece en el panel de control.

Si desea convertir un solo carácter en mayúsculas, después de inicializar o obtener, llame a este método. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Cadena S = 's';
        Edit1-> Text = S.UpperCase ();
}
// ---------------------------------------------- --------------------------
```

Puede utilizar este mismo método para convertir una cadena en mayúsculas de la siguiente manera:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Cadena S1 = "James N. Fame!";
        Cadena S2 = S1.UpperCase ();
        Edit1-> Text = S2;
}
// ---------------------------------------------- --------------------------
```

Además del método AnsiString, puede utilizar la función UpperCase () para convertir un carácter o una cadena en mayúsculas. Su sintaxis es:

```
AnsiString __fastcall UpperCase (const AnsiString S);
```

Esta función utiliza el mismo algoritmo que el método AnsiString :: UpperCase (). He aquí un ejemplo de cómo usarlo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Cadena S1 = "James N. Fame!";
        Cadena S2 = UpperCase (S1);
        Edit2-> Text = S2;
}
// ---------------------------------------------- --------------------------
```

La función AnsiUpperCase () utiliza la misma sintaxis que la función UpperCase () y aplica el mismo algoritmo que el método AnsiString :: UpperCase (). A diferencia de la función UpperCase (), AnsiUpperCase () considera la configuración regional de la computadora que se está utilizando. Observe cómo estas dos funciones convierten la misma cadena francesa:

```
// ---------------------------------------------- ---
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
    Cadena S1 = "La casa? Ça a été brûlée!";
    Cadena S2 = UpperCase (S1); Edit1-> Text = S2;
}
// ---------------------------------------------- ---
```


```
// ---------------------------------------------- ---
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
    Cadena S1 = "La casa? Ça a été brûlée!";
    Cadena S2 = AnsiUpperCase (S1);
    Edit1-> Text = S2;
}
// ---------------------------------------------- ---
```

Casos de cuerdas: Conversión de mayúsculas a minúsculas

Puede utilizar el **método AnsiString :: LowerCase ()** para convertir un carácter o una cadena en mayúsculas en minúsculas. Su sintaxis es:

```
AnsiString __fastcall LowerCase () const;
```

Utilizando la configuración regional, esta función examina cada carácter de la variable AnsiString proporcionada. Si un carácter es un carácter alfabético en mayúsculas, se convertiría en minúsculas. El caso de todos los demás caracteres sería ignorado.

Si desea convertir un solo carácter en mayúsculas, después de inicializar o obtener, llame a este método. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: btnConvertClick (TObject * Sender)
```

```
{
        String String1 = "[Borland C + + Builder]";
        Edit1-> Text = String1.LowerCase ();
}
// ---------------------------------------------- --------------------------
```

También puede utilizar la función LowerCase () para convertir un carácter o una cadena en minúsculas. Su sintaxis es:

```
AnsiString __fastcall LowerCase (const AnsiString S);
```

Esta función utiliza el mismo algoritmo que el **método AnsiString :: UpperCase ()** . He aquí un ejemplo de cómo usarlo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: btnConvertClick (TObject * Sender)
{
        String String1 = "[Borland C + + Builder]";
        Edit1-> Text = LowerCase (String1);
}
// ---------------------------------------------- --------------------------
```

Si la configuración local que está utilizando o la distribución de su programa a un factor, debe utilizar la función **AnsiLowerCase ()** . Procesa la cadena de la misma manera que el **método AnsiString :: UpperCase ()** pero utiliza la misma sintaxis que la función **UpperCase ()** :

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: btnConvertClick (TObject * Sender)
{
        String S1 = "La versión francesa de Borland C ++ Builder est là."
                    "Elle est arrivée!";
        EdtConvert-> Text = AnsiLowerCase (S1);
}
// ---------------------------------------------- --------------------------
```

Adición de Cuerdas

El Operador de Adición

Para agregar un valor AnsiString a otro, utilice el operador de adición (se sobrecargó para responder adecuadamente). La operación produciría una nueva cadena que combina la primera y la segunda cadena. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Edit3-> Text = Edit1-> Text + Edit2-> Texto;
}
// ---------------------------------------------- --------------------------
```

A continuación, puede utilizar la operación de suma para agregar tantas cuerdas como le parezca.

Adición de cadenas

Añadir dos cadenas consiste en agregar una cadena a otra. Esta operación normalmente implica dos cadenas: un destino y una fuente de cadenas. Para añadir dos cadenas, además del operador de adición "+", puede utilizar la función de miembro **AppendStr ()** . Su sintaxis es:

```
Void __fastcall AppendStr (AnsiString & Destination, const AnsiString Source);
```

Esta función toma dos argumentos como cadenas. La fuente se agrega al destino. La función devuelve el destino ya cambiado por la operación. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString Destination = "Paul";
        AnsiString Fuente = "Lombard";
        AppendStr (Destino, Origen);
        Edit1-> Text = Destino;
}
// ---------------------------------------------- --------------------------
```

Funciones de comparación de cadenas

Introducción

La comparación de cadenas le permite averiguar cuál de dos cadenas es más larga o si ambas cadenas son iguales. Al comparar dos cadenas, el compilador a veces comprueba caracteres en minúsculas y mayúsculas. Dependiendo de la función que esté utilizando, puede pedirle al compilador que considere el caso de los caracteres; Esto se conoce como caso-sensibilidad. Algunas de las funciones, al realizar su comparación en fechas o valores de moneda, se refieren a la configuración regional de su equipo como se establece en el panel de control.

Comparación de cadenas con case-insensitivity

La función **SameText ()** se utiliza para comparar dos cadenas. Su sintaxis es:

```
Bool __fastcall SameText (const AnsiString String1, const AnsiString String2);
```

La función requiere dos variables AnsiString y las compara. La comparación se realiza sin sensibilidad de mayúsculas y minúsculas. Si las cadenas son iguales, el resultado es verdadero (el equivalente entero es 1). De lo contrario, la comparación produce false (0). Puede utilizar la función SameText () en un diálogo de validación como éste:

A continuación, puede implementar el evento **OnClick ()** del botón Aceptar de la siguiente manera:

```
// ---------------------------------------------- --------------------------
Void __fastcall TOKBottomDlg :: OKBtnClick (TObject * Sender)
{
        Cadena Password1 = edtPassword1-> Text;
        String Password2 = edtPassword2-> Text;
        Resultado booleano = SameText (Password1, Password2);

        If (Resultado == Falso)
        {
                Panel1-> Caption = "¡Sus contraseñas no coinciden!";
                EdtPassword1-> SetFocus ();
        }
        más
        {
                Panel1-> Caption = "Su cuenta ha sido configurada.";
                Cerca();
        }
}
// ---------------------------------------------- --------------------------
```

El **AnsiString :: AnsiCompareIC ()** método realiza una comparación de dos cadenas, teniendo en cuenta la configuración regional. Al igual que **SameText ()** , esta función no se preocupa por la sensibilidad de mayúsculas y minúsculas

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString String1 = Editar1-> Texto;
        AnsiString String2 = Editar2-> Texto;

        If (String1.AnsiCompareIC (String2) <0)
                Edit3-> Text = "True";
        Else if (String1.AnsiCompareIC (String2)> 0)
                Edit3-> Text = "Falso";
        más
                Edit3-> Text = "Igual";
}
// ---------------------------------------------- --------------------------
```

Como alternativa, puede utilizar la función **CompareText ()** para comparar cadenas. A diferencia del **método AnsiString :: AnsiCompareIC ()** , esta función no se preocupa por la configuración regional de Windows. La sintaxis de esta función es:

```
Int __fastcall CompareText (const AnsiString Primero, const AnsiString Segundo);
```

La función toma dos argumentos de cadena y examina sus caracteres de forma incremental. Después de la comparación, la función devuelve:

- Un valor negativo si la primera cadena es menor que la segunda
- Un valor positivo si la primera cadena es mayor que la segunda
- 0 si ambas cadenas son iguales

Comparación de cadenas con case-sensitivity

El **método AnsiString :: AnsiCompare ()** se utiliza para comparar dos cadenas con respecto a la sensibilidad de mayúsculas y minúsculas. Esta función, cuando se realiza en fechas y valores de moneda, considera los ajustes regionales del equipo del usuario. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString String1 = Editar1-> Texto;
        AnsiString String2 = Editar2-> Texto;

        If (String1.AnsiCompare (String2) <0)
                Edit3-> Text = "True";
        Else if (String1.AnsiCompare (String2)> 0)
                Edit3-> Text = "Falso";
        más
                Edit3-> Text = "Igual";
}
// ---------------------------------------------- --------------------------
```

Además del **método AnsiString :: AnsiCompare ()** puede utilizar la función **AnsiCompareStr ()** para comparar cadenas. Al igual que el **método AnsiString :: AnsiCompare ()** , esta función toma en cuenta la configuración regional de Windows. Su sintaxis es:

```
Int __fastcall AnsiCompare (const AnsiString & OtherString ) const;
```

La función considera su propia cadena y la compara con el argumento de cadena que toma. Esta función devuelve:

- Un valor negativo si la cadena es menor que la *cadena OtherString*
- Un valor positivo si la cadena es mayor que la *cadena OtherString*
- 0 si ambas cadenas son iguales

Para comparar cadenas, también puede utilizar la función **CompareStr ()** . A diferencia del **método AnsiString :: AnsiCompare ()** , esta función no se preocupa por la configuración regional de Windows. La sintaxis de esta función es:

```
Int __fastcall CompareStr (const AnsiString Primero, const AnsiString Segundo);
```

La función toma dos argumentos de cadena y examina sus caracteres de forma incremental. Después de la comparación, la función devuelve:

- Un valor negativo si la primera cadena es menor que la segunda
- Un valor positivo si la primera cadena es mayor que la segunda
- 0 si ambas cadenas son iguales

Comparaciones booleanas de cadenas

La clase **AnsiString** y la biblioteca **sysutils** proporcionan técnicas de comparación de cadenas. Las funciones que hemos utilizado para realizar comparaciones devolvieron valores integrales al final de sus comparaciones. A veces, al realizar algoritmos específicos, como la comparación de contraseñas, realizar cálculos matemáticos, realizar comprobaciones de ortografía en documentos de texto, etc., sólo necesitará saber si dos cadenas son iguales. Este tipo de comparación convierte un valor booleano de verdadero o falso. Ambas bibliotecas pueden realizar cualquier tipo de comparación.

Cuando tiene dos cadenas y desea averiguar si ambos son iguales, puede utilizar el operador (sobrecargado) ==. Si ambas cadenas son iguales, la comparación condicional devolverá true.

También puede utilizar la función AnsiSameStr (). Su sintaxis es:

```
Bool __fastcall AnsiSameStr (const AnsiString Primero, const AnsiString Segundo);
```

La función toma en cuenta los ajustes regionales de Windows cuando se comparan las cadenas Primera y Secundaria con sensibilidad de mayúsculas y minúsculas. Si ambas cadenas son iguales, la función devuelve true. Si no son iguales, el resultado es falso. Aquí hay un ejemplo:

```
// ---------------------------------------------- ---------------------------
Void __fastcall TForm1 :: btnSendClick (TObject * Sender)
{
        String EMail1 = edtEMail1-> Texto;
        Cadena EMail2 = edtEMail2-> Texto;

        If (AnsiSameStr (EMail1, EMail2))
        {
                FrmCongratulaciones-> ShowModal ();
                Cerca();
        }
}
// ---------------------------------------------- ---------------------------
```

Como alternativa, para comparar dos cadenas, puede utilizar la función **AnsiSameText ()** . Ambas funciones utilizan la misma sintaxis. Al igual que la función **AnsiSameStr ()** , **AnsiSameText ()** considera la configuración regional. A diferencia de la función **AnsiSameStr ()** , la función **AnsiSameText ()** no considera el caso de los caracteres en las cadenas.

Si las cadenas que desea comparar son subtítulos, como los que ve en un formulario, sería engorroso escribir una función de comparación que los examinara. Esto se debe a que los subtítulos en un formulario suelen tener un signo y utilizado para subrayar uno de sus caracteres. Los ejemplos incluyen el nombre, la dirección, la ciudad, el nombre completo o el departamento. Por suerte, **Borland proporciona la función AnsiSameCaption ()** . Su sintaxis es:

```
Bool __fastcall AnsiSameCaption (const AnsiString Primero, const AnsiString Segundo);
```

Esta función toma dos subtítulos y los compara considerando los ajustes regionales. Independientemente del lugar en el que se coloque el esperma, se examinarían los demás caracteres de los subtítulos. Si ambos subtítulos son iguales, la función devolverá true. En el ejemplo siguiente, dos subtítulos se establecen como Nombre y Nombre respectivamente. Una comparación regular los encontraría diferentes, pero la función **AnsiSameCaption ()** encuentra que ambas cadenas son iguales:

```
// ---------------------------------------------- ---------------------------
Void __fastcall TForm1 :: FormCreate (TObject * Sender)
{
```

```
        LblCaption1-> Caption = "& Nombre";
        LblCaption2-> Caption = "Nombre y Nombre";
}
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: btnCompareCaptionsClick (TObject * Sender)
{
        Cadena Caption1 = lblCaption1-> Caption;
        Cadena Caption2 = lblCaption2-> Caption;

        If (AnsiSameCaption (Caption1, Caption2))
                Panel1-> Caption = "Mismas leyendas";
        más
                Panel1-> Caption = "Completamente diferente";
}
// ---------------------------------------------- --------------------------
```

Además de todos los métodos de comparación disponibles de la clase AnsiString y funciones de comparación a través de la VCL, puede utilizar los operadores booleanos para realizar cualquier tipo de comparaciones en cadenas. Estos operadores se pueden utilizar de la misma manera que se procedería con variables numéricas regulares. Por lo tanto, los operadores son:

- Igual
- No es igual:! =
- Menos de <
- Menos que o igual a <=
- Mayor que>
- Mayor o igual que> =

Caracteres y Sub-Strings

El último carácter de una cadena

Hay muchas operaciones que puede realizar en caracteres individuales de una variable AnsiString. Estos incluyen la comprobación de un carácter, la búsqueda de la posición de un carácter, o la eliminación de un carácter o caracteres. Estas operaciones pueden ser valiosas al crear objetos como cuadros de diálogo de inicio de sesión.

Para buscar el último carácter de una cadena, utilice el **método AnsiString :: AnsiLastChar ()** . Su sintaxis es:

```
Char * __fastcall AnsiLastChar () const;
```

Puede utilizar este método para averiguar el último carácter de una cadena dada. En el ejemplo siguiente, el último carácter de la cadena en el cuadro de edición Edit1 aparece en el cuadro de edición Edit2 cuando el usuario hace clic en el control Button1:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Cadena S = Editar1-> Texto;
        Edit2-> Text = S.AnsiLastChar ();
}
// ---------------------------------------------- --------------------------
```

Si el **AnsiString** se utiliza en una aplicación de consola, puede utilizar este mismo método para buscar el último carácter de una variable **AnsiString** .

Eliminación de caracteres

A veces usted querrá deshacerse de un personaje en una cadena. Esto se hace mediante el **método AnsiString :: Delete ()** . La sintaxis es:

```
AnsiString & __fastcall Delete (int Índice, int Cuenta);
```

Esta función miembro toma dos argumentos enteros. El primer argumento especifica la posición donde el compilador comenzaría considerando la eliminación. El segundo argumento especifica el número de caracteres que se deben eliminar de la variable AnsiString.

Si declara una variable AnsiString llamada País. Puede utilizar **Country.Delete (14, 11)** para eliminar 11 caracteres que comienzan en el carácter 14:

```
AnsiString Country ("Estados Unidos de América");
AnsiString Después;
Puts (Country.c_str ());
Después = Country.Delete (14, 11);
Puts (After.c_str ());
```

Sub-String Creación

Una subcadena es una cadena que se crea desde, o está incluida en, otra cadena. C ++ y C ++ Builder le permiten encontrar una subcadena en una cadena original, para obtener la posición de una subcadena en una cadena, etc.

Con una cadena, puede crear una nueva cadena recuperada desde el original utilizando la cadena AnsiString :: SubString (). Su sintaxis es:

```
AnsiString __fastcall SubString (int StartPosition, int HowManyChars) const;
```

Este método toma dos argumentos enteros. De la cadena original, el primer argumento especifica la posición del carácter donde se iniciaría la nueva cadena. El segundo argumento especifica el número de caracteres que se tendrían en cuenta al crear la nueva cadena. Aquí hay un ejemplo:

```
// ---------------------------------------------- ---
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        AnsiString Original = Editar1-> Texto;
        AnsiString SubOne = Original.SubString (9, 4);
        Edit2-> Text = SubOne;
}
// ---------------------------------------------- ---
```

La posición de una subcadena

La clase AnsiString le permite analizar una cadena y averiguar si contiene alguna subcadena. Si lo hace, puede obtener la posición de la subcadena, utilizando el AnsiString :: Pos () método. Su sintaxis es:

```
Int __fastcall Pos (const AnsiString & SubString) const;
```

En muchos casos, también puede utilizar el método AnsiString :: AnsiPos ()). Su sintaxis es:

```
Extern PACKAGE int __fastcall AnsiPos (constante AnsiString Substr, const AnsiString S);
```

Sustitución de caracteres o subcadenas

Al realizar sus algoritmos u otros tipos específicos de operaciones en cadenas, es posible que desee averiguar si un determinado carácter o grupo de caracteres se ha proporcionado en una cadena. Si es así, es posible que desee reemplazarlo por un carácter diferente o por una nueva subcadena. Esta operación es manejada por la función **StringReplace ()** . Su sintaxis es:

```
AnsiString __fastcall StringReplace (const AnsiString Original ,
                                     Const AnsiString LookFor ,
                                     Const AnsiString ReplaceWith ,
                                     TReplaceFlags FlagToUse );
```

La función **StringReplace ()** buscará el carácter *LookFor* o sub-cadena en la cadena original. Si lo encuentra, reemplazará el carácter LookFor o la *subcadena por el* carácter *ReplaceWith* o sub-cadena. Controla cómo se realiza esta operación utilizando el argumento *FlagToUse* . Los valores que puede utilizar son para reemplazar todas las apariciones de *LookFor* con *ReplaceWith* . El indicador usado sería *rfReplaceAll* . También puede pedirle al compilador que no tome en consideración el carácter (s) caso, que se hace con *rfIgnoreCase* . Una vez que el argumento *TReplaceFlags* es un conjunto, puede utilizar uno o ambos de los indicadores. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: Button1Click (TObject * Sender)
{
        Edit1-> Text = StringReplace (Edit1-> Text, "", "",
        TReplaceFlags () << rfReplaceAll);
}
// ---------------------------------------------- --------------------------
```

Citas de cuerdas

Conversión de cadenas regulares a cadenas entre comillas

En el estricto de las rutinas de cadena, una cita es un carácter o símbolo utilizado para delimitar una cadena. Establece el principio y el final de una cadena. En el idioma inglés, una cita se representa con el símbolo de comillas dobles ". El VCL está equipado con funciones utilizadas para insertar o quitar citas de una cadena.

La función **AnsiQuotedStr ()** se utiliza para "convertir" una cadena en una cadena entre comillas. Su sintaxis es:

```
AnsiString __fastcall AnsiQuotedStr (const AnsiString Fuente, char Cita);
```

Esta función toma una cadena, el argumento Source, y la devuelve agregó los caracteres Quote en ambos lados de la cadena. Aquí hay un ejemplo:


```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: edtQuotedExit (TObject * Sender)
{
        Char * BookTitle = edtQuoted-> Text.c_str ();
        Char Cita = '"';
        AnsiString Citado = AnsiQuotedStr (BookTitle, Cotización);
        EdtBookTitle-> Texto = Citado;
}
// ---------------------------------------------- --------------------------
```

Cadena entre comillas a la conversión de cadenas regulares

La función **QuotedStr ()** se utiliza para agregar una comilla única en ambos lados de una cadena. Su sintaxis es:

```
AnsiString __fastcall QuotedStr (const AnsiString S);
```

Esta función toma una cadena y la devuelve después de agregar una cita única en ambos lados. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: edtQuotedExit (TObject * Sender)
{
        Char * BookTitle = edtQuoted-> Text.c_str ();
        AnsiString Citado = QuotedStr (BookTitle);
        EdtBookTitle-> Texto = Citado;
}
// ---------------------------------------------- --------------------------
```

Eliminación de cotizaciones de cadenas

Cuando una cadena se proporciona con comillas y desea quitar las comillas, utilice la función AnsiExtractQuotedStr (). Su sintaxis es:

```
AnsiString __fastcall AnsiExtractQuotedStr (char * & Fuente, char Cita);
```

Esta función toma dos argumentos. El parámetro Source es una cadena terminada en null que se devuelve como un valor AnsiString. Al utilizar la función, debe especificar qué carácter o símbolo se utiliza como Cita. Aquí hay un ejemplo:

```
// ---------------------------------------------- --------------------------
Void __fastcall TForm1 :: edtQuotedExit (TObject * Sender)
{
        Char * BookTitle = edtQuoted-> Text.c_str ();
        Char Cita = '"';
        AnsiString RemoveQuotes = AnsiExtractQuotedStr (BookTitle, Cotización);
        EdtBookTitle-> Text = RemoveQuotes;
}
// ---------------------------------------------- --------------------------
```

