Manual técnico: 4 practica

Estudiante: Sebastian Zapata Alzate

Curso: lógica y representación I

Universidad de Antioquia

Facultad de ingeniería

Profesor: Oscar Lopera

Departamento: ingeniería de sistemas

Semestre: 2019-2

2020

Introducción

Este manual sirve como guía para explorar el aplicativo de la cuarta practica de laboratorio del curso de lógica y representación I de la facultad de ingeniería de la universidad de Antioquia. El manual ofrece puntos detallados de cómo funciona el código fuente en las partes del aplicativo.

Objetivo de la Practica

El objetivo de la practica es crear un programa por computadora, que permita generar un vector, el tamaño de este viene dado por el usuario, y a partir de ahí, el aplicativo debe cumplir efectivamente con ciertas funcionalidades:

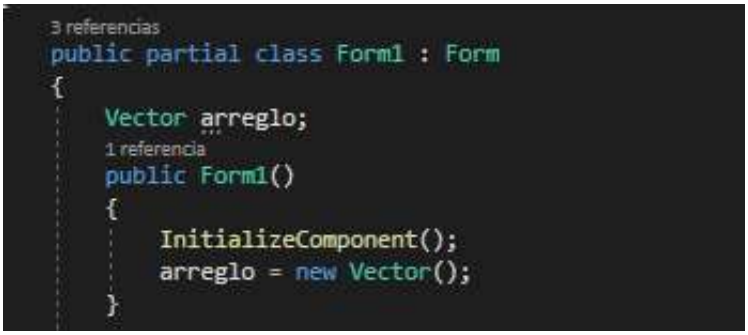
Funcionalidades:

1. El aplicativo debe permitirle al usuario ingresar el tamaño del vector, además le debe permitir si quiere llenar el vector con números aleatorios o si desea hacerlo manualmente
2. El programa le debe permitir al usuario cambiar un elemento del vector, basta con que el usuario ingrese la posición del elemento que quiere cambiar, y, tenga la posibilidad de ingresar un numero generado aleatoriamente o el mismo usuario ingresar el nuevo elemento.
3. El programa podrá cambiar de posición 2 elementos dentro del vector, para esto el usuario debe ingresar las posiciones de los elementos que desea intercambiar.
4. Por último, el aplicativo deberá poder invertir los elementos del vector si este así lo desea, para esto, deberá tener en cuenta que la enumeración de los arreglos en programación es distinta a la que la gente está acostumbrada.

   Extra: el aplicativo deberá tener en cuenta todo tipo de excepciones generada por el usuario en tiempo de ejecución, y controlarlas para que esto no afecte la ejecución correcta del aplicativo.

Implementación

1. En la primera parte se implementa el código para que el usuario pueda ingresar el tamaño del vector, y además se empiezan a considerar que excepciones pueda tener esa entrada dado el tipo de dato que necesitamos. Además se crea un objeto de la clase vector en la cual se implementan métodos asociados a vectores

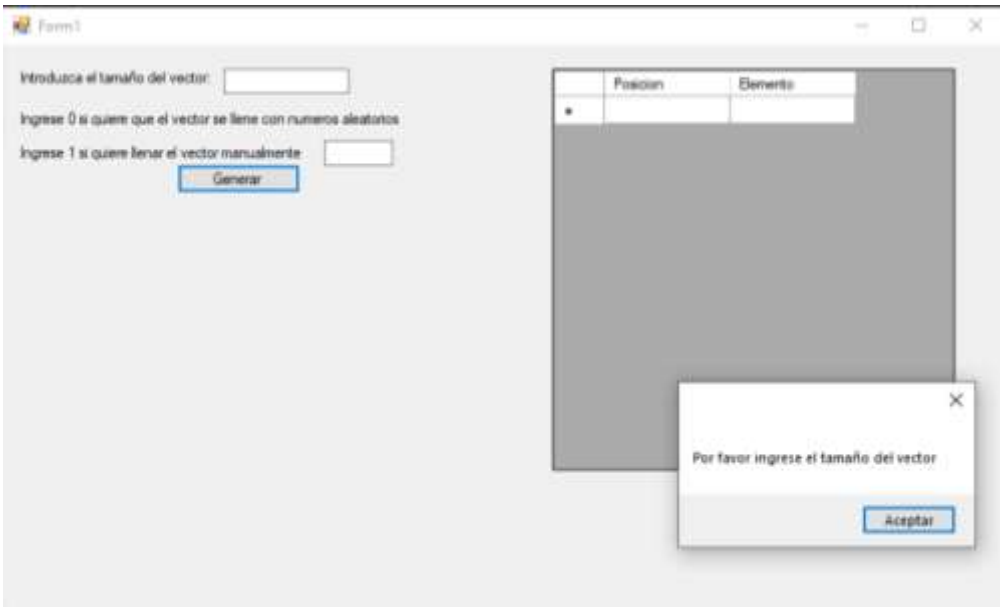


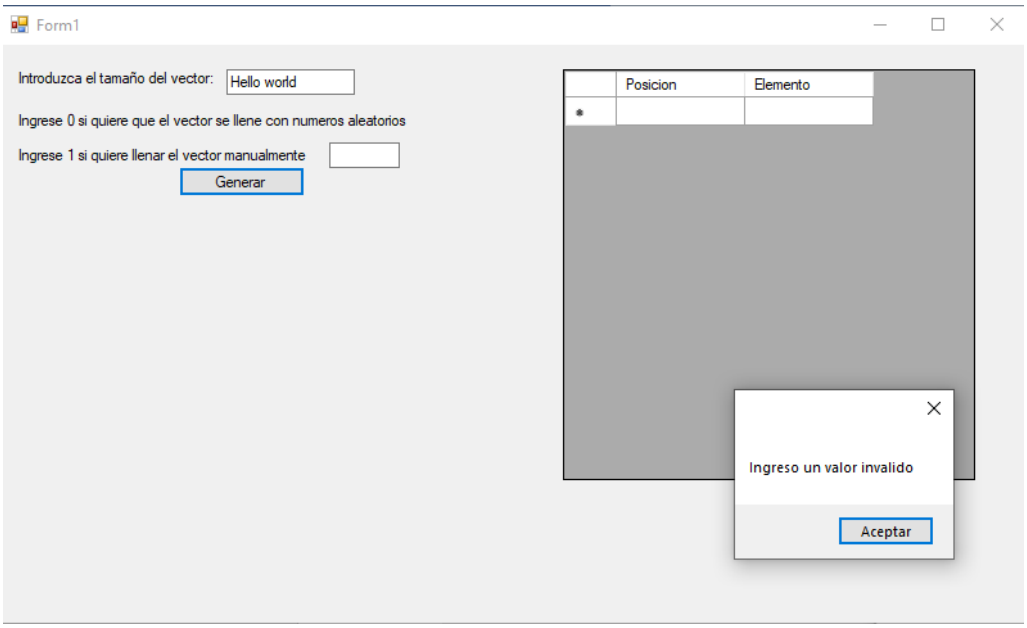
```
class Vector
{
    //Llena el vector con numeros aleatorios
    1 referencia
    public int[] llenar_Aleatorios(int[] vector, int x)
    {
        Random aleatorio = new Random();

        for (int i = 0; i < x; i++)
        {

            int numero_aleatorio = aleatorio.Next(0, 100);
            vector[i] = numero_aleatorio;
        }
        return vector;
    }
    //invierte los elementos de un vector
    1 referencia
    public int[] Invertir_vector(int[] vector, int x)
    {

        int aux;
        int auxiliar_invertir;
        int b = x;

        for (int i = 0; i <= b / 2; i++)
        {
            auxiliar_invertir = vector[i];
            vector[i] = vector[b - 1];
            vector[b - 1] = auxiliar_invertir;

            b--;
        }
        return vector;
    }
```

Clase vector que cuenta con 2 metodos, el primero recibe un vector y su tamaño y llena el vector con numeros aleatorios y devuelve el vector. El segundo, recibe un vector y su tamaño e invierte los elementos del vector, y lo devuelve invertido.

```
private void button1_Click(object sender, EventArgs e)
{
    if (textbox.Text == "") // Muestra un mensaje si no se ingresa un valor en la longitu
    {
        MessageBox.Show("Por favor ingrese el tamaño del vector");
        textbox.Focus();
        return;
    }

    int x; // variable donde se almacena la longitud del vector

    //Controla la excepcion de que el usuario no ingrese valores numericos
    try
    {
        x = Int32.Parse(textbox.Text);
    }
    catch (Exception d)
    {
        MessageBox.Show("Ingreso un valor invalido");
        textbox.Focus();
        return;
    }

    // Controla la excepcion de que el usuario ingrese un valor negativo
    if (x <= 0)
    {
        MessageBox.Show("No se puede ingresar un tamaño negativo para el vector");
        textbox.Focus();
        return;
    }


    int[] v = new int[x];
    int[] vector = v; // creamos el vector que va almacenar la informacion
```

En este fragmento se implementan 3 controles de excepciones en el método correspondiente al botón generar, la primera controla la restricción de que el usuario no ingrese ningún valor en el tamaño del vector y presione en generar, la segunda controla las excepciones de que ingrese letras (puesto que el tamaño es un numero entero) de igual forme restringe la entrada de números decimales, fraccionarios. El 3 controla el hecho de que el usuario ingrese un valor negativo para el tamaño. Además se crea un vector con el que se trabajara posteriormente.



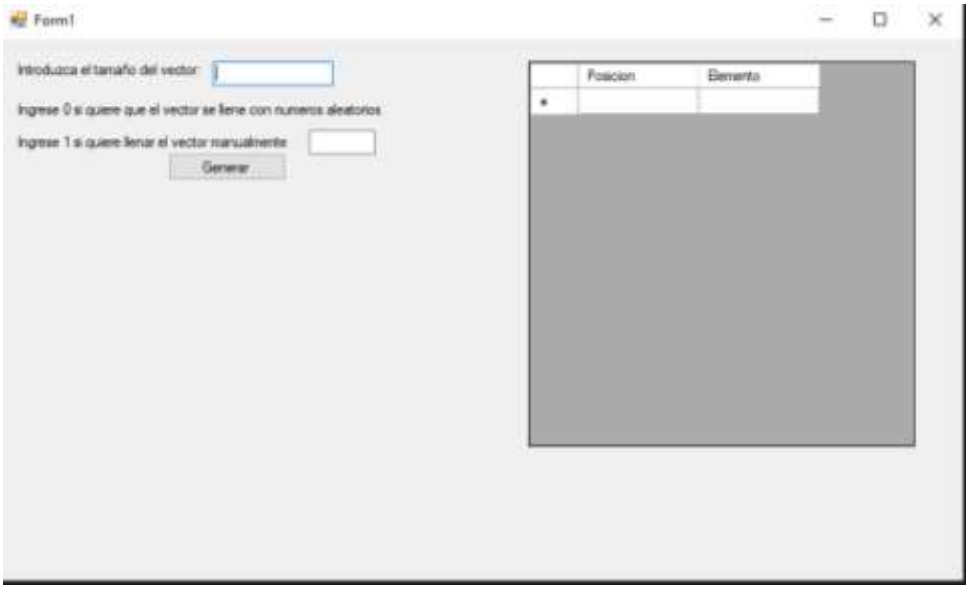


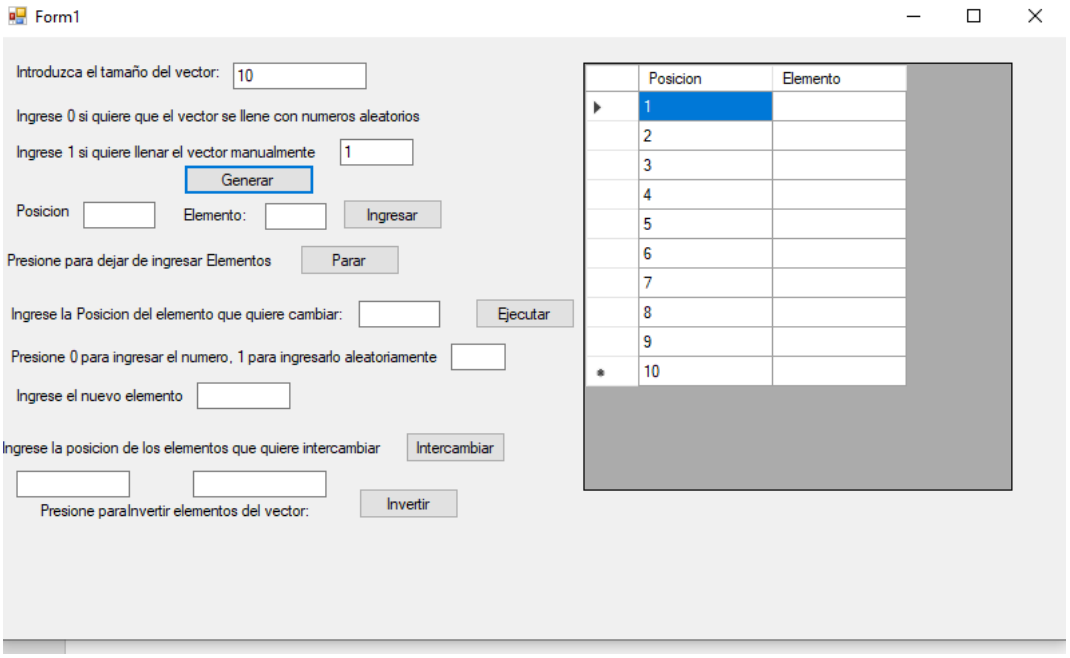
```
int[] v = new int[x];
int[] vector = v; // creamos el vector que va almacenar la informacion

//Determinar si el usuario quiere ingresar los elementos del vector
// o quiere que el programa los genere aleatoriamente

//Controlamos que el usuario no ingrese valor alguno en el textbox
if (txtdecision.Text == "")
{
    MessageBox.Show("Por favor ingrese como quiere que el vector se llene");
    txtdecision.Focus();
    return;
}

int decision; // variable donde se almacena la longitud del vector

//Controla la excepcion de que el usuario no ingrese un valor numerico
try
{
    decision = Int32.Parse(txtdecision.Text);
}
catch (Exception a)
{
    MessageBox.Show("Las posibilidades no incluyen letras o numeros diferentes de enteros");
    txtdecision.Focus();
    return;
}

// Controlamos si el usuario ingresa un valor distinto de 0  o 1
if (decision < 0 || decision > 1)
{
    MessageBox.Show("El valor ingresado esta por fuera de las posibilidades");
    txtdecision.Focus();
    return;
}
```

En este bloque, Se le pide al usuario que escoja si quiere llenar El vector aleatoriamente o si quiere ingresar los elementos por su propia cuenta, además se tienen en cuenta las restricciones que esta entrada pueda generar, casi iguales a las de la entrada del tamaño del vector.

```
//Si el usuario escoge la primera opcion, el vector se llena aleatoriamente
if (decision == 0)
{

    lista.Rows.Add(x - 1);

    arreglo.llenar_Aleatorios(vector, x); // Llenamos el vector con numeros aleatorios

    for (int i = 0; i < x; i++)
    {
        //agregamos los elementos de la primer columna

        lista.Rows[i].Cells[0].Value = i + 1;
        lista.Rows[i].Cells[1].Value = vector[i];
    }
}
//Algoritmo que se implementa si el usuario decide ingresar los valores manualmente
else if (decision == 1)
{
    // Mostramos los forms correspondientes a esta opcion

    lbelemento.Visible = true;
    txtelemento.Visible = true;
    btingresar.Visible = true;
    lbpos.Visible = true;
    txtpos.Visible = true;
    lbstop.Visible = true;
    Parar.Visible = true;
    lbintercambio.Visible = true;
    txtintercambio1.Visible = true;
    txtintercambio2.Visible = true;
    btintercambio.Visible = true;
    btinvert.Visible = true;
    lbinvert.Visible = true;
    lbcambio.Visible = true;
```

Aquí se implementan las 2 opciones de acuerdo a la decisión del usuario, si como dice la interfaz gráfica, el usuario ingresa 0, el vector se llena con números aleatorios, para esto se hace la llamada del método en la clase Vector y además se llenan las casillas de las 2 columnas del datagridview (control que permita visualizar el vector), en la ultima parte hay un bloque que el usuario le pudiera resultar curioso o particular y es una de las soluciones implementadas para que el usuario no genere excepciones, y es que se le ocultan las demás funcionalidades del programa hasta que el usuario no genere el vector con datos ingresados correctamente.





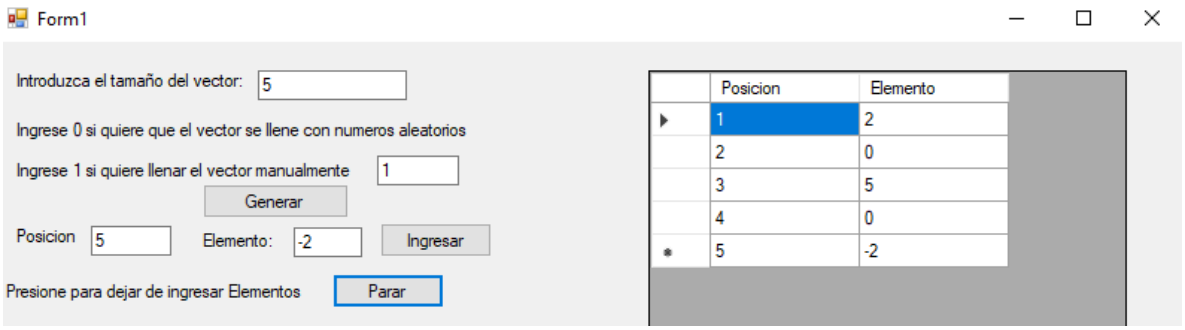
```
1 referencia
public void btingresar_Click(object sender, EventArgs e)
{
    int x = Int32.Parse(textbox.Text);
    int posicion;
    int elemento;

    if (txtpos.Text == "") // controlamos la excepcion de que no ingrese un valor
    {
        MessageBox.Show("Ingrese por favor una posicion");
        txtpos.Focus();
        return;
    }
    else
    {
        try
        {
            posicion = Int32.Parse(txtpos.Text); // intentamos convertirlo a tipo int
        }
        catch (Exception d)
        {
            MessageBox.Show("Por favor ingrese un valor numerico entero");
            txtpos.Focus();
            return;

        }
    }

    if (txtelemento.Text == "") // controlamos la excepcion de que no ingrese un valor
    {
        MessageBox.Show("Ingrese por favor un elemento");
        txtelemento.Focus();
        return;
    }
```

Aquí se empieza a implementar el código cuando el usuario decide ingresar los elementos del vector manualmente, aquí podemos ver como ose controlan las excepciones como en bloques pasados.

```
        if (posicion > x || posicion < 0)
        {
            MessageBox.Show("La posicion ingresada no existe en el vector");
        }
        else
        {
            lista.Rows[posicion - 1].Cells[1].Value = elemento;
        }

    }


    1 referencia
    private void Parar_Click(object sender, EventArgs e)
    {
        int x = Int32.Parse(textbox.Text);


        int[] vector = new int[x];

        //LLena el vector de ceros en caso de que el usuario no quiera seguir llenandolo
        for (int j = 0; j < x; j++)
        {
            if (lista.Rows[j].Cells[1].Value == null)
            {
                lista.Rows[j].Cells[1].Value = 0;
            }
        }
```

En este fragmento es donde el usuario puede ingresar una posicion y el elemento correspondiente a esa posicion, ademas, notese que el usuario puede ingresar elementos en posiciones distantes, esto es. Puede dejar vacios entre los elementos, a lo cual el programa los llena con ceros como se ve en la siguiente imagen.

```
1 referencia
private void Parar_Click(object sender, EventArgs e)
{
    int x = Int32.Parse(textbox.Text);


    int[] vector = new int[x];

    //LLena el vector de ceros en caso de que el usuario no quiera seguir llenandolo
    for (int j = 0; j < x; j++)
    {
        if (lista.Rows[j].Cells[1].Value == null)
        {
            lista.Rows[j].Cells[1].Value = 0;
        }
    }

}
```




```
1 referencia
private void btchange_Click(object sender, EventArgs e)
{
    int posicion_cambio;
    int x = Int32.Parse(textbox.Text);

    //Evaluamos si la posicion ingresada no genera excepciones

    if (txtcambio.Text == "")
    {
        MessageBox.Show("Por favor ingresar una posicion para cambiarle el valor");
        txtcambio.Focus();
        return;
    }
    else
    {
        try
        {
            posicion_cambio = Int32.Parse(txtcambio.Text);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Por favor ingrese un numero entero");
            txtcambio.Focus();
            return;
        }

    }

    //evaluamos si la posicion si se encuentra dentro del vector
    if (posicion_cambio > x || posicion_cambio < 0)
    {
        MessageBox.Show("la posicion no existe dentro del vector");
        txtcambio.Focus();
        return;
```

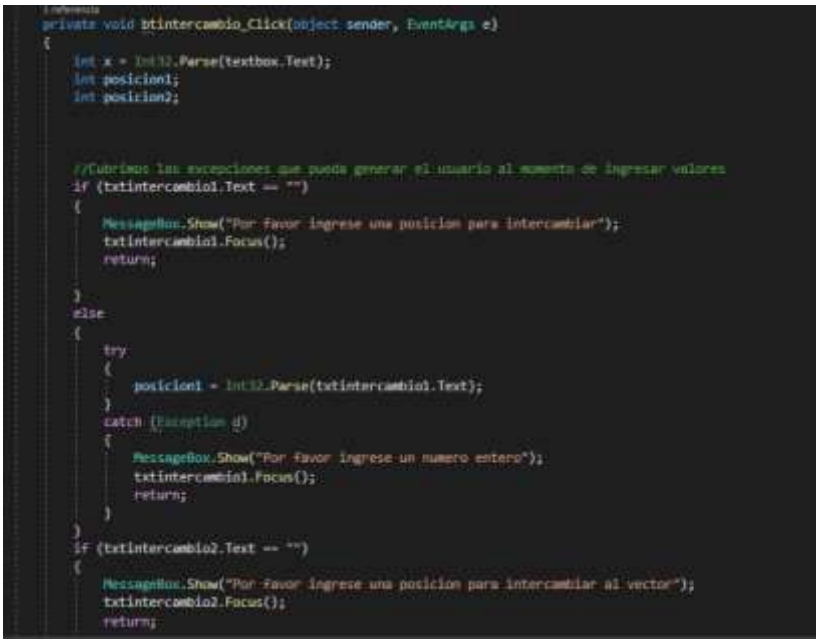
2. Ahora describiremos com el aplicativo cumple con el segundo punto, que es cambiar un elemento del vector por un numero ingresado por el usuario o generado aleatoiamente, en la imagen anterior se controlan las excepciones correspondientes.

```
// ahora evaluemos la decision tomada por el usuario de si ingresar un numero aleatorio
// o que el mismo lo ingrese

int choice;

if (txtchoice.Text == "")
{
    MessageBox.Show("Por favor escoga una opcion");
    txtchoice.Focus();
    return;
}
else
{
    try
    {
        choice = Int32.Parse(txtchoice.Text);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Por favor ingrese un numero entero");
        txtchoice.Focus();
        return;

    }

}
if (choice < 0 || choice > 1)
{
    MessageBox.Show("Escogio una posibilidad invalidad");
    txtchoice.Focus();
    return;

}
```

La variable choice, pertenece a la decision del usuario de si ingresar el elemento o generarlo aleatoriamente, como es una entrada del usuario podria generar excepciones en tiempo de ejecucion que debemos controlar, la ultima corresponde a si ingresa un numero distinto de cero o uno (que son las unicas posibilidades)

```
    }
    else if (choice == 0)
    {
        int elemento;

        if (txtchange.Text == "")
        {
            MessageBox.Show("Por favor ingres un elemento");
            txtchange.Focus();
            return;
        }
        else
        {
            elemento = Int32.Parse(txtchange.Text);
            lista.Rows[posicion_cambio - 1].Cells[1].Value = elemento;
        }
    }
    else if (choice == 1)
    {
        int elemento;

        Random aleatorio = new Random();
        elemento = aleatorio.Next(0, 100);

      lista.Rows[posicion_cambio - 1].Cells[1].Value = elemento;


    }
}

1 reference
private void btintercambio_Click(object sender, EventArgs e)
{
    int x = Int32.Parse(textbox.Text);
```

Despues de controlar las excepciones, ya se cambia el elemento; si la decision es 1 entonces lo genera aleatoriamente, y si es cero lo ingresa en tiempo de ejecucion.



```
private void btintercambio_Click(object sender, EventArgs e)
{
    int x = Int32.Parse(textbox.Text);
    int posicion1;
    int posicion2;


    //Cubrimos las excepciones que pueda generar el usuario al momento de ingresar valores
    if (txtintercambio1.Text == "")
    {
        MessageBox.Show("Por favor ingrese una posicion para intercambiar");
        txtintercambio1.Focus();
        return;

    }
    else
    {
        try
        {
            posicion1 = Int32.Parse(txtintercambio1.Text);
        }
        catch (Exception d)
        {
            MessageBox.Show("Por favor ingrese un numero entero");
            txtintercambio1.Focus();
            return;
        }
    }
    if (txtintercambio2.Text == "")
    {
        MessageBox.Show("Por favor ingrese una posicion para intercambiar al vector");
        txtintercambio2.Focus();
        return;
```

3. En este fragmento del codigo fuente, empezamos a dar forma al 3 punto de intercambiar de posicion elementos existentes en el vector, como hemos podido ver a lo largo del programa, las excepciones no se dejan esperar.

```
    }

    //Realizamos el intercambio si la posicion ingresada es valida
    else

    {
        int auxiliar;

        auxiliar = Int32.Parse(lista.Rows[posicion1 - 1].Cells[1].Value.ToString());
        lista.Rows[posicion1 - 1].Cells[1].Value = lista.Rows[posicion2 - 1].Cells[1].Value;

        lista.Rows[posicion2 - 1].Cells[1].Value = auxiliar;
    }


}
```

Aquí vemos el algoritmo que permite intercambiar los elementos del vector, es necesario resaltar la importancia de la variable auxiliar ya que si no se usa, solo se daria un intercambio porque como se hace primero un intercambio, ya el valor anterior que necesitabamos poner en la otra posicion es el que intercambiamos, pero si lo almacenamos en una variable auxiliar lo podemos guardar y usar.

```
1 referencia
private void btInvert_Click(object sender, EventArgs e)
{
    int x = Int32.Parse(textbox.Text); //almacenamos la longitud del vector

    int[] vector = new int[x]; // creamos un vector que almacena la segunda columna del datagrid
    int aux;
    int auxiliar_invertir;
    int j = x;

    //creamos el bucle for que recorre los elementos de la segunda columna
    //y los almacena en un arreglo

    for (int i = 0; i < x; i++)
    {
        aux = Int32.Parse(lista.Rows[i].Cells[1].Value.ToString());
        vector[i] = aux;
    }


    arreglo.Invertir_vector(vector, x);

    //agregamos una columna al nuevo datagrid
    lista2.Rows.Add(x - 1);

    //hacemos visible en el form el nuevo datagrid que muestra el vector invertido
    lista2.Visible = true;

    for (int j = 0; j < x; j++)
    {
        lista2.Rows[j].Cells[0].Value = j + 1; //mostramos el nuevo vector con sus posiciones
        lista2.Rows[j].Cells[1].Value = vector[j];
    }

}
```

4. Por ultimo, el programa permite invertir los elementos del vector, para esto usamos un metodo de la clase vector, primero debemos recorrer la segunda columna que corresponde a los elementos, al almacenarlo en un arreglo, podemos usar el metodo de la clase vector para invertirlos, despues activamos otro datagrid que habia permanecido oculto y este permite mostrar el vector invertido, por ultimo lo mostramos en este nuevo datagridview.