

UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE ELECTROTECNIA Y COMPUTACIÓN

Algoritmización y Estructuras de Datos

Profesor:

Adilson G. López

Grupo: 2M1 - CO

Desarrolladores:

Gabriel A. Ortiz — 2020 - 0325U

Marcel E. Díaz — 2020 - 1384U

Índice

1. Introducción	2
2. Código Fuente	4
2.1. Clases	4
2.1.1. Estudiante «Student»	4
2.1.2. Empleado «Employee»	4
2.2. Formulario MDI Principal	5
2.2.1. Diseño:	5
2.2.2. Propiedades:	5
2.2.3. Declaraciones globales e inicializaciones:	6
2.2.4. Métodos de interacción con la interfaz	7
2.3. Formulario Registro Acerca de	8
2.3.1. Diseño:	8
2.3.2. Propiedades:	8
2.4. Formulario Registro Académico	9
2.4.1. Diseño:	9
2.4.2. Métodos de la clase	10
2.4.3. Métodos de ordenamiento	14
2.4.4. Eventos de la clase	18
2.5. Formulario Registro Laboral	23
2.5.1. Diseño:	23
2.5.2. Métodos de la clase	24
2.5.3. Métodos de ordenamiento	27
2.5.4. Eventos de la clase	31
3. Conclusión (Ejecución del programa)	36
3.1. Formulario «Acerca de»	36
3.2. Formulario «Registro Laboral»	37
3.3. Formulario «Registro Académico»	38

1. Introducción

En este laboratorio se estará tratando registro de datos por medio de arreglos desordenados, con un sencillo CRUD que lleva los siguientes elemntos:

1. Carnet
2. Nombre y Apellido
3. I Semestre
4. II Semestre
5. Nota Final

y otro con los siguientes atributos:

1. Cedula
2. Nombre
3. Apellido
4. Salario
5. N° Hijos

Las herramientas con las que se estará trabajando son:

1. TextBox
2. DataGridView
3. ComboBox
4. Labels
5. Buttons
6. TableLayoutPanel
7. PictureBox
8. Panel

El lenguaje utilizado para este laboratorio es C#, lenguaje creado por Microsoft. Utilizamos en su misma medida .NET Framework con el IDE de Visual Studio 2019. Con el fin de aprender las herramientas que posee tanto el lenguaje con el entorno de desarrollo a la vez que se aprenden distintos algoritmos de manejo de datos.



2. Código Fuente

2.1. Clases

2.1.1. Estudiante «Student»

Esta clase se emplea en el formulario correspondiente al registro académico, en la cual se almacenará la información de los estudiantes a ingresar.

```
● ● ●
1  namespace AP1.Models
2  {
3      class Student
4      {
5          public string IdCard { get; set; }
6          public string Name { get; set; }
7          public string LastName { get; set; }
8          public int FirstPartial { get; set; }
9          public int SecondPartial { get; set; }
10         public int Systematic { get; set; }
11         public double FinalNote { get; set; }
12     }
13 }
```

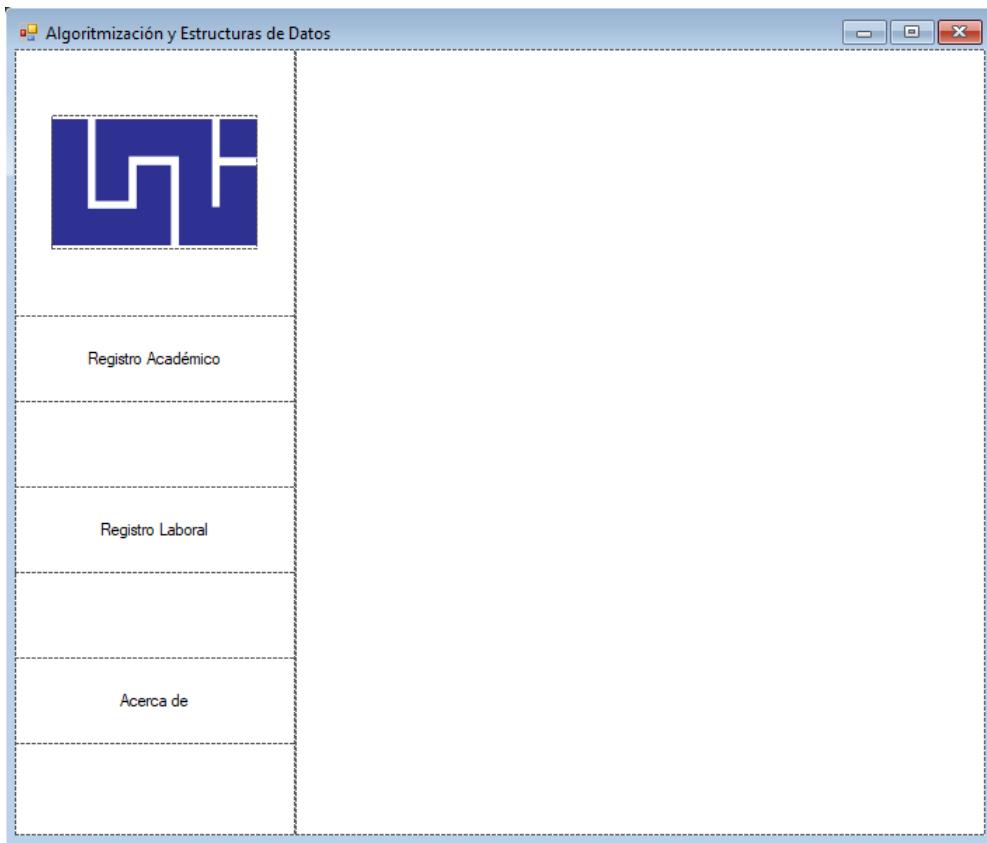
2.1.2. Empleado «Employee»

Paralelamente, esta clase se emplea en el formulario correspondiente al registro laboral.

```
● ● ●
1  namespace AP1.Models
2  {
3      class Employee
4      {
5          public string IdentificationCard { get; set; }
6          public string Names { get; set; }
7          public string LastNames { get; set; }
8          public double Salary { get; set; }
9          public double Bonus { get; set; }
10         public double NetSalary { get; set; }
11         public int Sons { get; set; }
12     }
13 }
14 }
15 }
```

2.2. Formulario MDI Principal

2.2.1. Diseño:



2.2.2. Propiedades:

1. Name: MdiMain
2. Minimum Size: 620x500
3. Size: 700x600
4. StartPosition: CentreScreen
5. Name: Algoritmización y estructura de datos

2.2.3. Declaraciones globales e inicializaciones:

Como C# trabaja con programación orientada a objetos, cada uno de los formularios nos genera una clase que declararemos e instanciaremos como global en nuestro formulario principal. Al momento de cargar nuestro formulario, le enviamos nuestras instancias de formularios a un panel contenedor ubicado a la derecha de nuestro form. Luego, utilizamos el método Show para visualizar el formulario llamado frmAbout cuando se inicialice nuestro formulario principal:

```
● ● ●
1 FrmAbout frmAbout = new FrmAbout();
2 FrmAcademyRecord frmAcademyRecord = new FrmAcademyRecord();
3 FrmEmploymentRecord frmEmploymentRecord = new FrmEmploymentRecord();
4 public MdiMain()
5 {
6     InitializeComponent();
7 }
8 private void MdiMain_Load(object sender, EventArgs e)
9 {
10    ContainerPanel.Controls.Add(frmAbout);
11    ContainerPanel.Controls.Add(frmEmploymentRecord);
12    ContainerPanel.Controls.Add(frmAcademyRecord);
13    frmAbout.Show();
14 }
```

2.2.4. Métodos de interacción con la interfaz

Cada uno de los botones tiene la misma lógica de funcionalidad: llama al formulario correspondiente y oculta al resto. En el caso particular del formulario de información, oculta el logo de la UNI en el menú de la interfaz.

1. Botón Registro Académico:

```
● ● ●
1 private void BtnFrmAcademyRecord_Click(object sender, EventArgs e)
2 {
3     pbUni.Visible = true;
4     frmAbout.Hide();
5     frmEmploymentRecord.Hide();
6     frmAcademyRecord.Show();
7 }
```

2. Botón Registro Laboral:

```
● ● ●
1 private void BtnEmploymentRecord_Click(object sender, EventArgs e)
2 {
3     pbUni.Visible = true;
4     frmAbout.Hide();
5     frmEmploymentRecord.Show();
6     frmAcademyRecord.Hide();
7 }
```

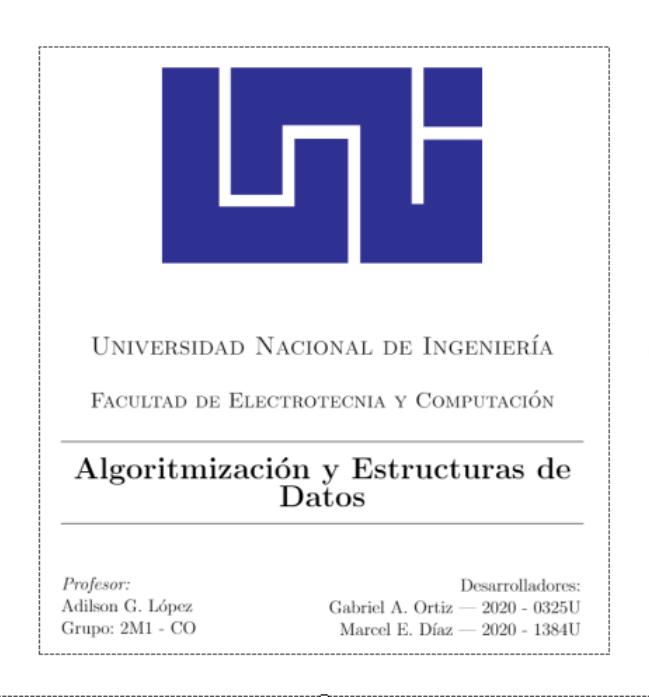
3. Botón Acerca de:

```
● ● ●
1 private void BtnAbout_Click(object sender, EventArgs e)
2 {
3     pbUni.Visible = false;
4     frmAbout.Show();
5     frmEmploymentRecord.Hide();
6     frmAcademyRecord.Hide();
7 }
```

2.3. Formulario Registro Acerca de

En este panel se muestra la información completa de los desarrolladores. Para ello, fue empleada una PictureBox en la que se cargó el archivo PNG de dicha información.

2.3.1. Diseño:



2.3.2. Propiedades:

1. Nombre: FrmAbout
2. BackColor: White
3. ControlBox: False
4. FormBorderStyle: None
5. TopLevel: False
6. Dock: Fill

2.4. Formulario Registro Académico

2.4.1. Diseño:

Inicializa un arreglo de tipo «Student» –clase previamente descrita– sin dimensionar llamado database. De igual forma, cuenta con una variable contador denominado students (con ‘s’ minúscula).

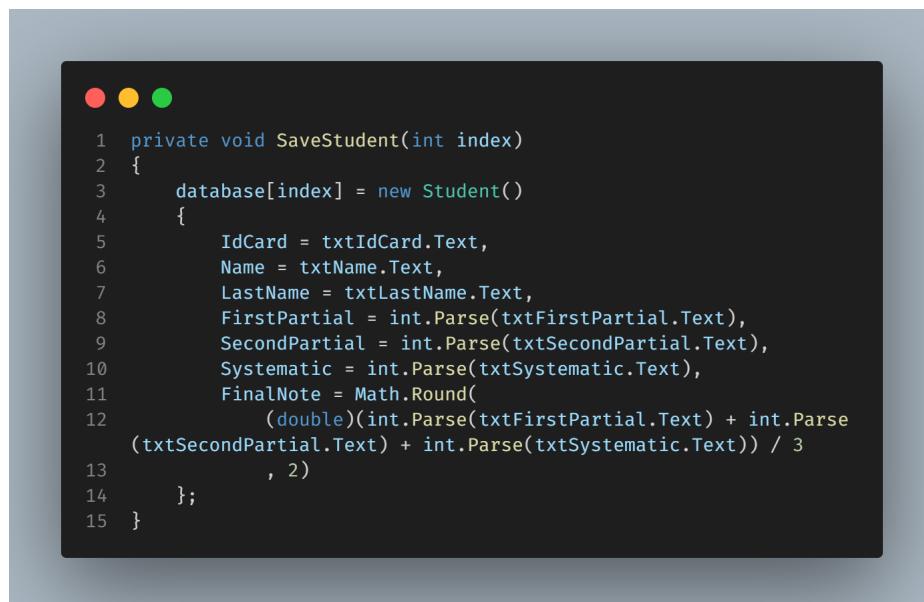
```

    ● ○ ●
 1  private Student[] database;
 2  private int students = 0;
 3  public FrmAcademyRecord()
 4  {
 5      InitializeComponent();
 6      TopLevel = false;
 7      Dock = DockStyle.Fill;
 8  }

```

2.4.2. Métodos de la clase

SaveStudent: Recibe un parámetro como índice que será empleado para guardar en el espacio del arreglo database la información de una nueva instancia Student.



The screenshot shows a code editor window with a dark theme. At the top left, there are three colored circular icons: red, yellow, and green. The code itself is written in C# and defines a private void method named SaveStudent. This method takes an integer parameter index and uses it to access a database array. It then initializes a new Student object and sets its properties based on text input from several text boxes: IdCard, Name, LastName, FirstPartial, SecondPartial, Systematic, and FinalNote. The FinalNote is calculated by summing the FirstPartial and SecondPartial values, dividing the result by 3, and rounding it to two decimal places. Finally, the method returns.

```
1  private void SaveStudent(int index)
2  {
3      database[index] = new Student()
4      {
5          IdCard = txtIdCard.Text,
6          Name = txtName.Text,
7          LastName = txtLastName.Text,
8          FirstPartial = int.Parse(txtFirstPartial.Text),
9          SecondPartial = int.Parse(txtSecondPartial.Text),
10         Systematic = int.Parse(txtSystematic.Text),
11         FinalNote = Math.Round(
12             (double)(int.Parse(txtFirstPartial.Text) + int.Parse
13             (txtSecondPartial.Text) + int.Parse(txtSystematic.Text)) / 3
14             , 2)
15     };
16 }
```

ValidateData: Es una función que retorna un booleano. Esta se cerciora de que todos los campos estén rellenados con la información correcta. De no ser así, retorna un falso.



The screenshot shows a Windows application window with a dark theme. At the top left, there are three colored circles (red, yellow, green). The main area contains a code editor with the following C# code:

```
1 private bool ValidateData()
2 {
3     try
4     {
5         // Validar campos
6         if (string.IsNullOrEmpty(txtIdCard.Text) ||
7             string.IsNullOrEmpty(txtName.Text) ||
8             string.IsNullOrEmpty(txtLastName.Text) ||
9             string.IsNullOrEmpty(txtFirstPartial.Text) ||
10            string.IsNullOrEmpty(txtSecondPartial.Text) ||
11            string.IsNullOrEmpty(txtSystematic.Text))
12             throw new Exception("Debe llenar todos los valores para ingresar un nuevo estudiante");
13
14         // Validar enteros en las calificaciones
15         if (!txtSecondPartial.Text.All(char.IsNumber) ||
16             !txtFirstPartial.Text.All(char.IsNumber) ||
17             !txtSystematic.Text.All(char.IsNumber)
18             )
19             throw new Exception("Debe ingresar un número para las calificaciones");
20
21         return true;
22     }
23     catch(Exception ex)
24     {
25         MessageBox.Show(ex.Message, "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Warning);
26         return false;
27     }
28 }
```

Clean: Vacía todos los campos del formulario.



```
1 private void Clean()
2 {
3     txtIdCard.Clear();
4     txtName.Clear();
5     txtLastName.Clear();
6     txtFirstPartial.Clear();
7     txtSecondPartial.Clear();
8     txtSystematic.Clear();
9     btnUpdate.Enabled = false;
10    btnDelete.Enabled = false;
11    btnInsert.Enabled = true;
12 }
```

GetIndexByIdCard: En una variable idCard almacena el texto ingresado en la caja de texto txtIdCard. Entonces, recorre todo el arreglo database en búsqueda de una coincidencia. De encontrarla, retorna la posición de la coincidencia en el arreglo. Por defecto, retornará -1.



```
1 private int GetIndexByIdCard()
2 {
3     string idCard = txtIdCard.Text;
4
5     for(int i = 0; i < students; i++)
6         if (database[i].IdCard == idCard)
7             return i;
8     return -1;
9 }
```

CalculateStats: Esta función declara una variable de tipo Student que albergará la información del mejor estudiante, se iniciará dependiendo si el arreglo tiene un elemento en el espacio 0 sino instanciará la clase Student. También se declara la variable double promedio y se inicia en 0. Recorrerá todos los elementos del arreglo en busca de la nota más alta a la par de ir sumando el promedio a la variable anteriormente mencionada. Para finalizar imprime los resultados en el label que contiene el promedio del salón y al mejor estudiante.

```
● ● ●
1  private void CalculateStats()
2  {
3      Student bestStudent = database[0] == null ? new Student() :
database[0];
4      double average = 0;
5      for (int i = 0; i < students; i++)
6      {
7          if (i < students - 1)
8              if (database[i].FinalNote > database[i + 1].FinalNot
e)
9                  bestStudent = database[i];
10             else
11                 bestStudent = database[i + 1];
12
13             average += database[i].FinalNote / students;
14         }
15
16         lblAverage.Text = "Promedio del salón: " + average;
17         lblBestStudent.Text = $"Mejor estudiante: {bestStudent.Name
} {bestStudent.LastName}";
18     }
```

2.4.3. Métodos de ordenamiento

Intercambiar posición:

```
● ● ●  
1 private void InvertPosition(int pos1, int pos2)  
2 {  
3     var temp = database[pos1];  
4     database[pos1] = database[pos2];  
5     database[pos2] = temp;  
6 }
```

Burbuja Mayor:

```
● ● ●  
1 private void MajorBubble(bool upward = true)  
2 {  
3     for (int i = students - 1; i > 0; i--)  
4     {  
5         for (int j = 0; j < i; j++)  
6         {  
7             if ((database[j].FinalNote > database[j + 1].FinalNote  
8 && upward) || (database[j].FinalNote < database[j + 1].FinalNote  
9 && !upward))  
10            {  
11                InvertPosition(j, j + 1);  
12            }  
13        }  
14    }  
15 }
```

Burbuja Menor:

```
● ● ●  
1 private void MinorBubble()  
2 {  
3     for (int i = 1; i < students; i++)  
4     {  
5         for (int j = students - 1; j > 0; j--)  
6         {  
7             if (database[j - 1].FinalNote > database[j].FinalNote)  
8             {  
9                 InvertPosition(j, j - 1);  
10            }  
11        }  
12    }  
13 }
```

Burbuja con Señal:

```
● ● ●  
1 private void SignBubble()  
2 {  
3     bool isFinish = false;  
4     for (int i = 0; i < students - 1 && isFinish == false; i++)  
5     {  
6         isFinish = true;  
7         for (int j = 0; j < students - 1; j++)  
8         {  
9             if (database[j].FinalNote > database[j + 1].FinalNote)  
10             {  
11                 InvertPosition(j, j + 1);  
12                 isFinish = false;  
13             }  
14         }  
15     }  
16 }
```

Sacudida:

```
● ● ●  
1  private void ShakerSort()  
2  {  
3      int left = 1, right = students - 1, k = right;  
4  
5      while (right ≥ left)  
6      {  
7          for (int i = right; i ≥ left; i--)  
8          {  
9              if (database[i - 1].FinalNote > database[i].FinalNote)  
10                 {  
11                     InvertPosition(i - 1, i);  
12                     k = i;  
13                 }  
14             }  
15             left = k + 1;  
16             for (int i = left; i ≤ right; i++)  
17             {  
18                 if (database[i - 1].FinalNote > database[i].FinalNote)  
19                 {  
20                     InvertPosition(i - 1, i);  
21                     k = i;  
22                 }  
23             }  
24             right = k - 1;  
25         }  
26     }
```

Inserción Directa:

```
● ● ●  
1  private void DirectInsert()  
2  {  
3      for (int i = 1; i < students; i++)  
4      {  
5          var temp = database[i];  
6          int k = i - 1;  
7  
8          while ((k ≥ 0) && (temp.FinalNote < database[k].FinalNote  
    ))  
9          {  
10             database[k + 1] = database[k];  
11             k--;  
12         }  
13         database[k + 1] = temp;  
14     }  
15 }
```

Inserción Indirecta:

```
● ● ●  
1  private void Shell()  
2  {  
3      bool hasChange;  
4      int inta = students;  
5  
6      while (inta > 0)  
7      {  
8          inta = (int)(inta / 2);  
9          hasChange = true;  
10         while (hasChange)  
11         {  
12             hasChange = false;  
13             for (int i = 0; (i + inta) ≤ (students - 1); i++)  
14             {  
15                 if (database[i].FinalNote > database[i + inta].Fin  
alNote)  
16                 {  
17                     InvertPosition(i, i + inta);  
18                     hasChange = true;  
19                 }  
20             }  
21         }  
22     }  
23 }
```

2.4.4. Eventos de la clase

Al cambiar método de ordenamiento: Al cambiar el método de ordenamiento del ComboBox llamado CbOrder, los datos en pantalla se mostrarán de forma ascendente o descendente, de forma correspondiente:



```
1  private void CbOrder_SelectedIndexChanged(object sender, EventArgs e)
2  {
3      try
4      {
5          if (cbOrder.SelectedIndex == 0)
6              MajorBubble(false);
7          else
8              MajorBubble();
9
10         dgvStudents.DataSource = database.ToList();
11     }
12     catch
13     {
14         MessageBox.Show("Ha ocurrido un error inesperado, contactarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
15     }
16 }
```

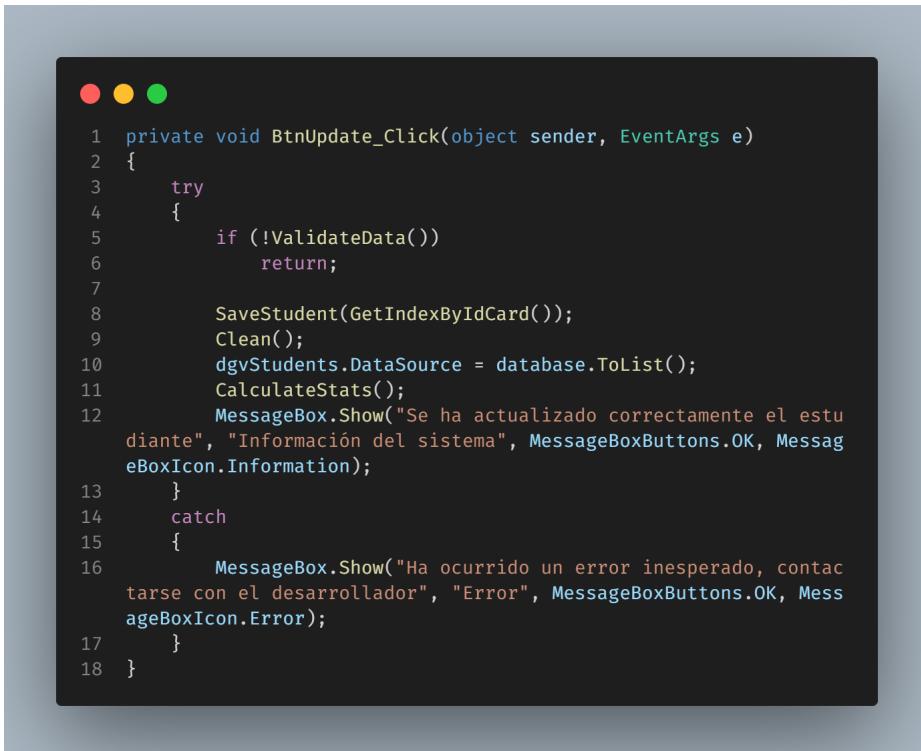
Click – Botón Insertar: Intenta realizar un bloque de código en el cual se valida la función ValidateData y redimensiona el arreglo database. Guarda al estudiante en la posición del arreglo actual mediante SaveStudent. Entonces, limpia todas las cajas de texto, incrementa el contador students y muestra un mensaje de notificación. De encontrar un error, no efectúa el bloque de código anterior y notifica el error.



The screenshot shows a Windows application window with a dark theme. At the top, there are three colored circles (red, yellow, green) which are standard window control buttons. Below them is a code editor window containing the following C# code:

```
1  private void BtnInsert_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          if (!ValidateData())
6              return;
7
8          Array.Resize(ref database, students + 1);
9
10         SaveStudent(students);
11
12         Clean();
13         dgvStudents.DataSource = database.ToList();
14         students++;
15         CalculateStats();
16         MessageBox.Show("Se ha ingresado correctamente el estudiante", "Información del sistema", MessageBoxButtons.OK, MessageBoxIcon.Information);
17     }
18     catch
19     {
20         MessageBox.Show("Ha ocurrido un error inesperado, contactarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
21     }
22 }
```

Click – Botón Actualizar: Del mismo modo que el botón insertar, intenta validar que todos los campos estén rellenos con la función ValidateData. Entonces, actualiza al estudiante en su posición correspondiente en el arreglo database. Vuelve a calcular las calificaciones y muestra la notificación pertinente. De encontrar un error, se notifica.



The screenshot shows a Windows application window with a dark theme. In the top-left corner, there are three colored circles (red, yellow, green) which are standard window control buttons. The main area of the window contains a code editor with C# code. The code is for a button click event named `BtnUpdate_Click`. It includes logic to validate data, save the student, calculate statistics, and show a message box if successful or an error if there is one. The code is numbered from 1 to 18.

```
1  private void BtnUpdate_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          if (!ValidateData())
6              return;
7
8          SaveStudent(GetIndexByIdCard());
9          Clean();
10         dgvStudents.DataSource = database.ToList();
11         CalculateStats();
12         MessageBox.Show("Se ha actualizado correctamente el estu-
diante", "Información del sistema", MessageBoxButtons.OK, MessageBoxIcon.Information);
13     }
14     catch
15     {
16         MessageBox.Show("Ha ocurrido un error inesperado, contac-
tarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
17     }
18 }
```

Click – Botón Eliminar: Intenta realizar un bloque de código; en él, se llama a la función GetIndexByIdCard dentro de una variable de iteración del ciclo for. Inmediatamente luego de encontrada una coincidencia, procede a trasladar los elementos posteriores al estudiante a eliminar un índice antes. Decrementa el contador students y en la última posición del arreglo database se instancia un nuevo estudiante.



The screenshot shows a Windows application window with three circular icons at the top left (red, yellow, green). The main area contains the following C# code:

```
1  private void BtnDelete_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          for (int i = GetIndexByIdCard(); i < students - 1; i++)
6              database[i] = database[i + 1];
7
8          students--;
9          database[students] = new Student();
10
11         Clean();
12         dgvStudents.DataSource = database.ToList();
13         CalculateStats();
14         MessageBox.Show("Se ha eliminado correctamente el estudiante", "Información del sistema", MessageBoxButtons.OK, MessageBoxIcon.Information);
15     }
16     catch
17     {
18         MessageBox.Show("Ha ocurrido un error inesperado, contactarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
19     }
20 }
21 }
```

Al soltar una tecla – casilla carnet: Cada vez que se suelta una tecla en la caja de texto txtIdCard, se llama a la función GetIndexByIdCard en búsqueda de una coincidencia; si la encuentra, imprime la información de dicho estudiante en cada una de los campos. Finalmente, activa los botones Modificar y Eliminar y desactivar Insertar.



The screenshot shows a Windows application window with a dark theme. In the top-left corner, there are three colored circles (red, yellow, green) which are standard window control buttons. The main area of the window contains a code editor with C# code. The code is a method named `TxtIdCard_KeyUp` with parameters `object sender, KeyEventArgs e`. The code logic involves getting an index from a function `GetIndexByIdCard`, then setting three buttons' enabled states based on whether an index was found (true for update/delete, false for insert). It also updates four text boxes (`txtName`, `txtLastName`, `txtFirstPartial`, `txtSecondPartial`) with student information from a database array. Finally, it toggles the enable state of the buttons again based on whether an index was found.

```
1 private void TxtIdCard_KeyUp(object sender, KeyEventArgs e)
2 {
3     int index = GetIndexByIdCard();
4     if (index >= 0)
5     {
6         btnUpdate.Enabled = true;
7         btnDelete.Enabled = true;
8         btnInsert.Enabled = false;
9         txtName.Text = database[index].Name;
10        txtLastName.Text = database[index].LastName;
11        txtFirstPartial.Text = database[index].FirstPartial.ToString();
12        txtSecondPartial.Text = database[index].SecondPartial.ToString();
13        txtSystematic.Text = database[index].Systematic.ToString();
14    }
15    else
16    {
17        btnUpdate.Enabled = false;
18        btnDelete.Enabled = false;
19        btnInsert.Enabled = true;
20    }
21 }
```

2.5. Formulario Registro Laboral

2.5.1. Diseño:

The diagram illustrates the design of the Employee Registration Form. It features a top section labeled "Datos del empleado" containing five input fields: "Cédula", "Nombres", "Apellidos", "Salario", and "Hijos". To the right of this section is a vertical toolbar with three buttons: "Insertar", "Actualizar", and "Eliminar". Below the input fields is a table view with columns: Cédula, Nombres, Apellidos, Hijos, Salario, Bono, and Salario Neto. The table has one row of data.

Inicializa un arreglo de tipo «Employee» –clase previamente descrita– sin dimensionar llamado database. De igual forma, cuenta con una variable contadora denominado employees (con ‘s’ minúscula).

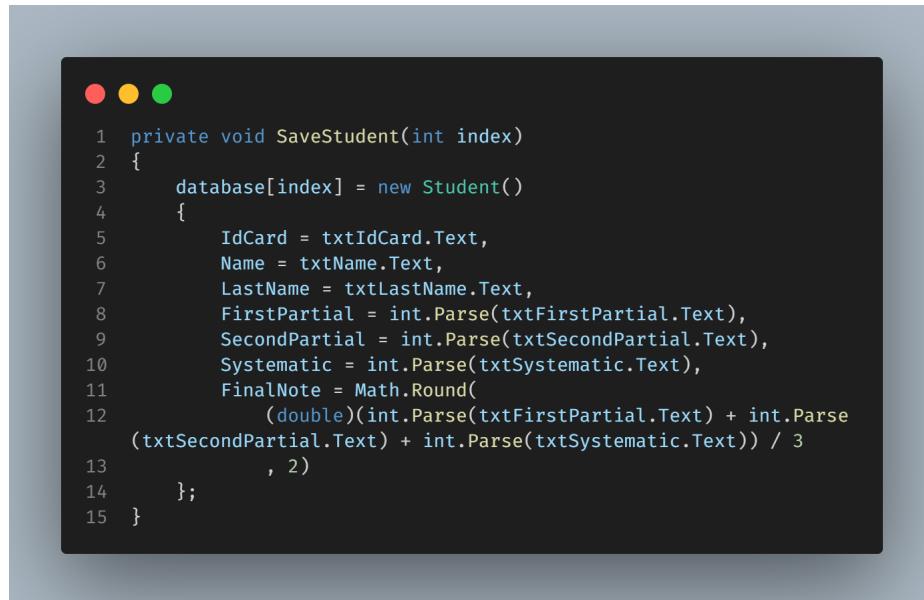
```

    ● ○ ●
1  private Employee[] database;
2  private int employees = 0;
3  public FrmEmploymentRecord()
4  {
5      InitializeComponent();
6      TopLevel = false;
7      Dock = DockStyle.Fill;
8  }

```

2.5.2. Métodos de la clase

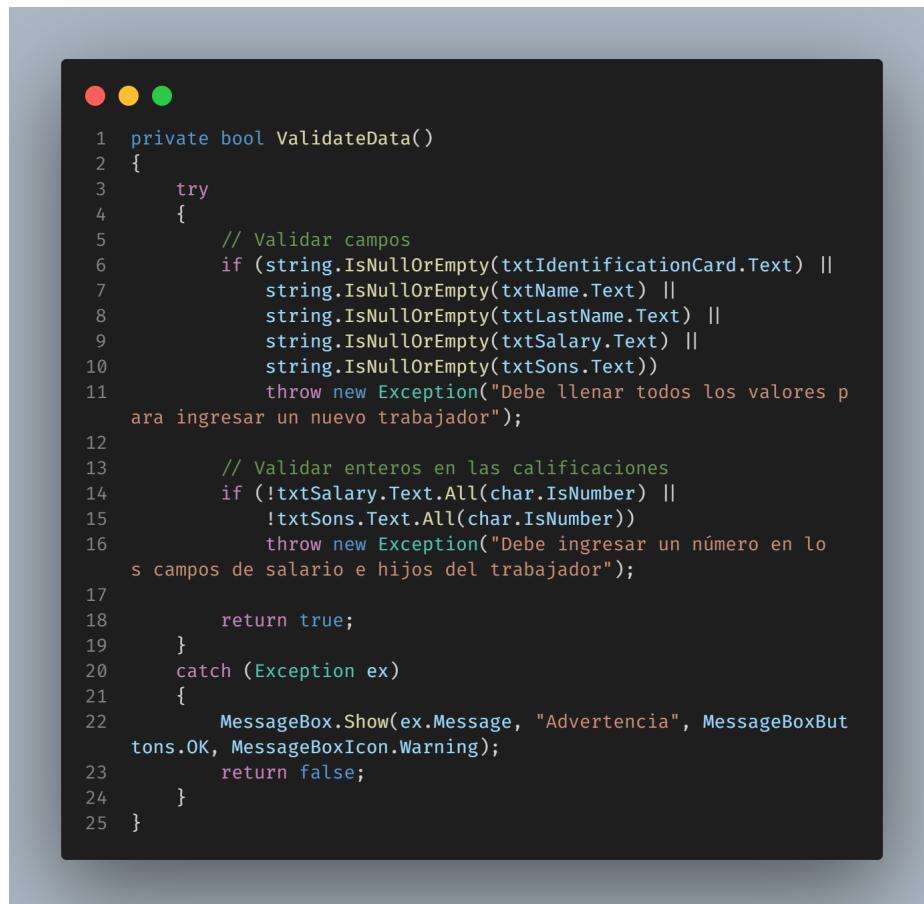
SaveEmployee: Recibe un parámetro como índice que será empleado para guardar en el espacio del arreglo database la información de una nueva instancia Employee.



The screenshot shows a code editor window with a dark theme. At the top left, there are three colored circular icons: red, yellow, and green. Below them, the code for the `SaveStudent` method is displayed:

```
1  private void SaveStudent(int index)
2  {
3      database[index] = new Student()
4      {
5          IdCard = txtIdCard.Text,
6          Name = txtName.Text,
7          LastName = txtLastName.Text,
8          FirstPartial = int.Parse(txtFirstPartial.Text),
9          SecondPartial = int.Parse(txtSecondPartial.Text),
10         Systematic = int.Parse(txtSystematic.Text),
11         FinalNote = Math.Round(
12             (double)(int.Parse(txtFirstPartial.Text) + int.Parse
13             (txtSecondPartial.Text) + int.Parse(txtSystematic.Text)) / 3
14             , 2)
15     };
16 }
```

ValidateData: Es una función que retorna un booleano. Esta se cerciora de que todos los campos estén rellenados con la información correcta. De no ser así, retorna un falso.



The screenshot shows a Windows application window with three circular icons at the top left (red, yellow, green). The main area contains the following C# code:

```
1 private bool ValidateData()
2 {
3     try
4     {
5         // Validar campos
6         if (string.IsNullOrEmpty(txtIdentificationCard.Text) ||
7             string.IsNullOrEmpty(txtName.Text) ||
8             string.IsNullOrEmpty(txtLastName.Text) ||
9             string.IsNullOrEmpty(txtSalary.Text) ||
10            string.IsNullOrEmpty(txtSons.Text))
11             throw new Exception("Debe llenar todos los valores para ingresar un nuevo trabajador");
12
13         // Validar enteros en las calificaciones
14         if (!txtSalary.Text.All(char.IsNumber) ||
15             !txtSons.Text.All(char.IsNumber))
16             throw new Exception("Debe ingresar un número en los campos de salario e hijos del trabajador");
17
18         return true;
19     }
20     catch (Exception ex)
21     {
22         MessageBox.Show(ex.Message, "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Warning);
23         return false;
24     }
25 }
```

Clean: Vacía todos los campos del formulario.

```
● ● ●  
1  private void Clean()  
2  {  
3      txtIdentificationCard.Clear();  
4      txtLastName.Clear();  
5      txtName.Clear();  
6      txtSalary.Clear();  
7      txtSons.Clear();  
8      btnUpdate.Enabled = false;  
9      btnDelete.Enabled = false;  
10     btnInsert.Enabled = true;  
11 }
```

GetIndexByIdCard: En una variable idCard almacena el texto ingresado en la caja de texto txtIdentificationCard. Entonces, recorre todo el arreglo database en búsqueda de una coincidencia. De encontrarla, retorna la posición de la coincidencia en el arreglo. Por defecto, retornará -1.

```
● ● ●  
1  private int GetIndexByIdCard()  
2  {  
3      string idCard = txtIdentificationCard.Text;  
4  
5      for (int i = 0; i < employees; i++)  
6          if (database[i].IdentificationCard == idCard)  
7              return i;  
8      return -1;  
9 }
```

2.5.3. Métodos de ordenamiento

Intercambiar posición:

```
● ● ●  
1 private void InvertPosition(int pos1, int pos2)  
2 {  
3     var temp = database[pos1];  
4     database[pos1] = database[pos2];  
5     database[pos2] = temp;  
6 }
```

Burbuja Mayor:

```
● ● ●  
1 private void MajorBubble(bool upward = true)  
2 {  
3     for (int i = employees - 1; i > 0; i--)  
4     {  
5         for (int j = 0; j < i; j++)  
6         {  
7             if ((database[j].Salary > database[j + 1].Salary && upward)  
8                 || (database[j].Salary < database[j + 1].Salary && !upward))  
9                 {  
10                     InvertPosition(j, j + 1);  
11                 }  
12             }  
13 }
```

Burbuja Menor:

```
● ● ●  
1 private void MinorBubble()  
2 {  
3     for (int i = 1; i < employees; i++)  
4     {  
5         for (int j = employees - 1; j > 0; j--)  
6         {  
7             if (database[j - 1].Salary > database[j].Salary)  
8             {  
9                 InvertPosition(j, j - 1);  
10            }  
11        }  
12    }  
13 }
```

Burbuja con Señal:

```
● ● ●  
1 private void SignBubble()  
2 {  
3     bool isFinish = false;  
4     for (int i = 0; i < employees - 1 && isFinish == false; i++)  
5     {  
6         isFinish = true;  
7         for (int j = 0; j < employees - 1; j++)  
8         {  
9             if (database[j].Salary > database[j + 1].Salary)  
10             {  
11                 InvertPosition(j, j + 1);  
12                 isFinish = false;  
13             }  
14         }  
15     }  
16 }
```

Sacudida:

```
● ● ●  
1  private void ShakerSort()  
2  {  
3      int left = 1, right = employees - 1, k = right;  
4  
5      while (right >= left)  
6      {  
7          for (int i = right; i >= left; i--)  
8          {  
9              if (database[i - 1].Salary > database[i].Salary)  
10                 {  
11                     InvertPosition(i - 1, i);  
12                     k = i;  
13                 }  
14             }  
15             left = k + 1;  
16             for (int i = left; i <= right; i++)  
17             {  
18                 if (database[i - 1].Salary > database[i].Salary)  
19                 {  
20                     InvertPosition(i - 1, i);  
21                     k = i;  
22                 }  
23             }  
24             right = k - 1;  
25         }  
26     }
```

Inserción Directa:

```
● ● ●  
1  private void DirectInsert()  
2  {  
3      for (int i = 1; i < employees; i++)  
4      {  
5          var temp = database[i];  
6          int k = i - 1;  
7  
8          while ((k >= 0) && (temp.Salary < database[k].Salary))  
9          {  
10              database[k + 1] = database[k];  
11              k--;  
12          }  
13          database[k + 1] = temp;  
14      }  
15  }
```

Inserción Indirecta:

```
● ● ●  
1  private void Shell()  
2  {  
3      bool hasChange;  
4      int inta = employees;  
5  
6      while (inta > 0)  
7      {  
8          inta = (int)(inta / 2);  
9          hasChange = true;  
10         while (hasChange)  
11         {  
12             hasChange = false;  
13             for (int i = 0; (i + inta) ≤ (employees - 1); i++)  
14             {  
15                 if (database[i].Salary > database[i + inta].Salary)  
16                 {  
17                     InvertPosition(i, i + inta);  
18                     hasChange = true;  
19                 }  
20             }  
21         }  
22     }  
23 }
```

2.5.4. Eventos de la clase

Click – Botón Insertar: Intenta realizar un bloque de código en el cual se valida la función ValidateData y redimensiona el arreglo database. Guarda al trabajador en la posición del arreglo actual mediante SaveEmployee. Entonces, limpia todas las cajas de texto, incrementa el contador employees y muestra un mensaje de notificación. De encontrar un error, no efectúa el bloque de código anterior y notifica el error.



The screenshot shows a Windows application window with three circular icons at the top left (red, yellow, green). The main area contains the following C# code:

```
1  private void BtnInsert_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          if (!ValidateData())
6              return;
7
8          Array.Resize(ref database, employees + 1);
9
10         SaveEmployee(employees);
11         employees++;
12         CalculateBonus();
13         dgvEmployee.DataSource = database.ToList();
14         MessageBox.Show("Se ha ingresado correctamente el trabajador", "Información del sistema", MessageBoxButtons.OK, MessageBoxIcon.Information);
15     }
16     catch
17     {
18         MessageBox.Show("Ha ocurrido un error inesperado, contactarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
19     }
20 }
```

Click – Botón Actualizar: Del mismo modo que el botón insertar, intenta validar que todos los campos estén llenados con la función ValidateData. Entonces, actualiza al estudiante en su posición correspondiente en el arreglo database. De encontrar un error, se notifica.



The screenshot shows a Windows application window with a dark theme. At the top left, there are three colored circles (red, yellow, green). Below them is a code editor containing the following C# code:

```
1  private void BtnUpdate_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          if (!ValidateData())
6              return;
7
8          SaveEmployee(GetIndexByIdCard());
9          CalculateBonus();
10         dgvEmployee.DataSource = database.ToList();
11         MessageBox.Show("Se ha actualizado correctamente el trabajador", "Información del sistema", MessageBoxButtons.OK, MessageBoxIcon.Information);
12     }
13     catch
14     {
15         MessageBox.Show("Ha ocurrido un error inesperado, contactarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
16     }
17 }
```

Click – Botón Eliminar: Intenta realizar un bloque de código; en él, se llama a la función GetIndexByIdCard dentro de una variable de iteración del ciclo for. Inmediatamente luego de encontrada una coincidencia, procede a trasladar los elementos posteriores al estudiante a eliminar un índice antes. Decrementa el contador employees y en la última posición del arreglo database se instancia un nuevo estudiante.



The screenshot shows a Windows application window with a dark theme. At the top left, there are three colored circles (red, yellow, green). Below them is a code editor containing the following C# code:

```
1  private void BtnDelete_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          for (int i = GetIndexByIdCard(); i < employees - 1; i++)
6              database[i] = database[i + 1];
7
8          employees--;
9          database[employees] = new Employee();
10         CalculateBonus();
11
12         dgvEmployee.DataSource = database.ToList();
13         MessageBox.Show("Se ha eliminado correctamente el trabajador", "Información del sistema", MessageBoxButtons.OK, MessageBoxIcon.Information);
14     }
15     catch
16     {
17         MessageBox.Show("Ha ocurrido un error inesperado, contactarse con el desarrollador", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
18     }
19 }
```

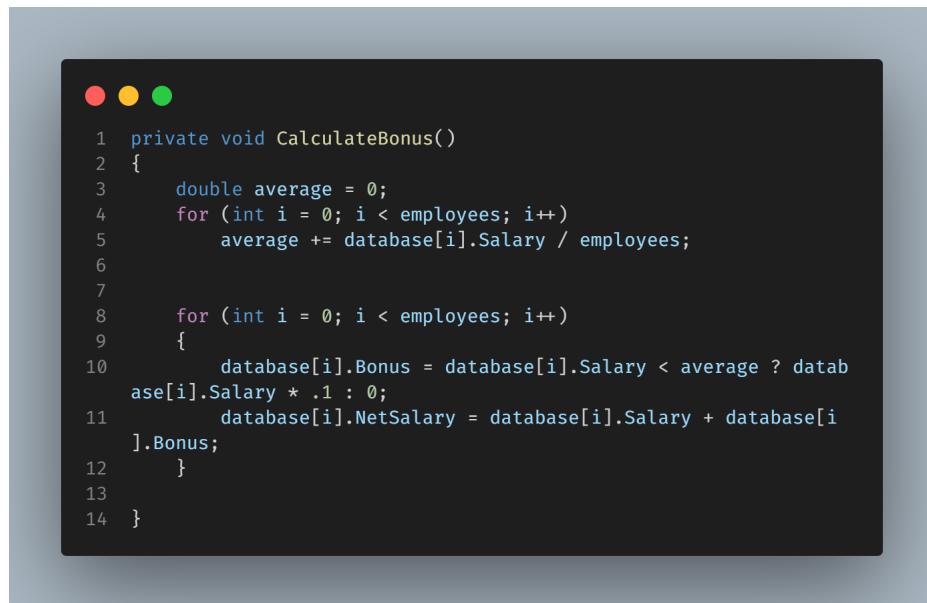
Al soltar una tecla – casilla cédula: Cada vez que se suelta una tecla en la caja de texto txtIdCard, se llama a la función GetIndexByIdCard en búsqueda de una coincidencia; si la encuentra, imprime la información de dicho trabajador en cada una de los campos. Finalmente, activa los botones Modificar y Eliminar y desactivar Insertar.



The screenshot shows a Windows application window with a dark theme. In the top-left corner, there are three colored circles (red, yellow, green) which are standard window control buttons. The main area of the window contains a code editor with C# code. The code is a method named `private void TxtIdentificationCard_KeyUp(object sender, EventArgs e)`. It uses the `GetIndexByIdCard()` function to get the index of the selected item in a list. If an index is found (greater than or equal to 0), it sets the enabled state of three buttons: `btnUpdate`, `btnDelete`, and `btnInsert`. It also updates four text boxes (`txtName`, `txtLastName`, `txtSalary`, `txtSons`) with the corresponding data from the database at the specified index. If no index is found (the `else` block), it disables `btnUpdate` and `btnDelete` and enables `btnInsert`.

```
1 private void TxtIdentificationCard_KeyUp(object sender, EventArgs e)
2 {
3     int index = GetIndexByIdCard();
4     if (index >= 0)
5     {
6         btnUpdate.Enabled = true;
7         btnDelete.Enabled = true;
8         btnInsert.Enabled = false;
9         txtName.Text = database[index].Names;
10        txtLastName.Text = database[index].LastNames;
11        txtSalary.Text = database[index].Salary.ToString();
12        txtSons.Text = database[index].Sons.ToString();
13    }
14    else
15    {
16        btnUpdate.Enabled = false;
17        btnDelete.Enabled = false;
18        btnInsert.Enabled = true;
19    }
20 }
```

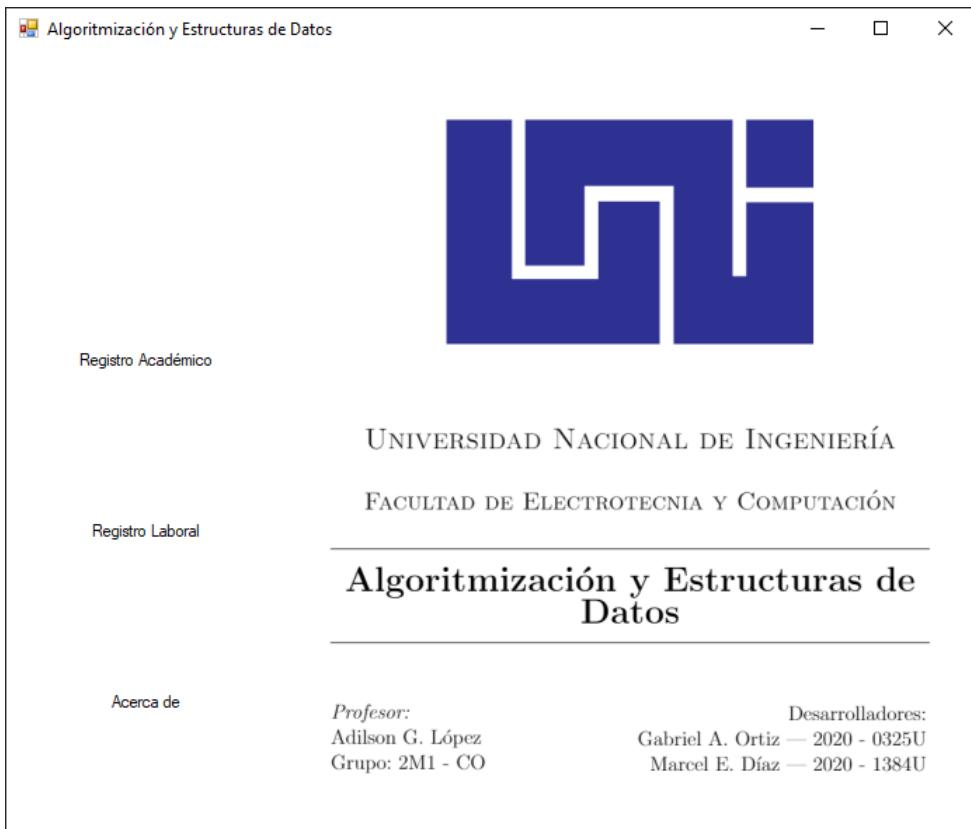
CalculateBonus: La función se encarga de calcular el salario promedio de todos los trabajadores y posterior a eso comparar que trabajadores ganan menos que el promedio para activar su bonus del 10 % sobre el salario real del mismo.



```
1  private void CalculateBonus()
2  {
3      double average = 0;
4      for (int i = 0; i < employees; i++)
5          average += database[i].Salary / employees;
6
7      for (int i = 0; i < employees; i++)
8      {
9          if (database[i].Salary < average)
10             database[i].Bonus = database[i].Salary * .1;
11             database[i].NetSalary = database[i].Salary + database[i]
12                 .Bonus;
13     }
14 }
```

3. Conclusión (Ejecución del programa)

3.1. Formulario «Acerca de»



3.2. Formulario «Registro Laboral»

Algoritmización y Estructuras de Datos

Datos del empleado

Cédula	Nombres	Apellidos	Hijos	Salario	Bono	Salario Neto

Insertar

Actualizar

Eliminar

Registro Académico

Registro Laboral

Acerca de

3.3. Formulario «Registro Académico»

The screenshot shows a Windows application window titled "Algoritmización y Estructuras de Datos". The main title bar has a logo consisting of blue letters "LGI". The window is divided into several sections:

- Registro Académico:** This section contains fields for "Camet" (text input), "Nombres" (text input), and "Apellidos" (text input). It also includes buttons for "Insertar" (Insert), "Modificar" (Modify), and "Eliminar" (Delete).
- Calificaciones:** This section contains fields for "I Parcial" (text input), "II Parcial" (text input), and "Sistemático" (text input).
- Ordenar:** A dropdown menu labeled "Ordenar" is located here.
- Registro Laboral:** A table with columns: Camet, Nombres, Apellidos, Primer Parcial, Segundo Parcial, and Sistemático. The table currently has one empty row.
- Acerca de:** A link or section name.
- Promedio de clase:** A label indicating the average grade of the class.
- Mejor estudiante:** A label indicating the best student.