UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA

UNAN–LEON

FACULTAD DE CIENCIAS Y TECNOLOGIA

CIENCIAS ACTUARIALES Y FINANCIERAS

CREACIÓN DE UNA APLICACIÓN EN EL PROGRAMADOR DE MICROSOFT EXCEL 2007 PARA LA ELABORACIÓN DE LA PLANILLA DE PAGOS DE UNA EMPRESA

PARA OPTAR AL TITULO DE
LICENCIADO EN CIENCIAS ACTUARIALES Y FINANCIERAS

Tutor:

Lic. Martín José Alonso Calderón

Autores:

- *Br. Freddy Neftalí Pineda Madrigal*
- *Br. José Adrián Bonilla Rivera*
- *Br. Nelson Enrique Jarquín Canales*

León, 30 de octubre del 2010

DEDICATORIA

"A mis padres y hermanos quienes siempre me han apoyado en mis decisiones y proyectos y quienes han sido mi inspiración para seguir adelante"
Freddy Neftalí Pineda Madrigal

"*A mis padres por su apoyo incondicional, porque gracias a su sacrificio pude formarme como profesional, a ellos dedico este trabajo*"
José Adrián Bonilla Rivera

"*A mi madre, quien me ha apoyado a lo largo de mi vida y es mi motivación para alcanzar mis metas*"
Nelson Enrique Jarquín Canales

AGRADECIMIENTO

Queremos agradecer a quienes han hecho posible la culminación de este proyecto, con el cual finaliza nuestra carrera e inicia una nueva etapa en nuestras vidas.

Primeramente a Jehová nuestro padre celestial quien es dador de vida y sabiduría, ya que gracias a su voluntad hemos culminado una etapa más de nuestra vida, a él gracias por sus bendiciones.

Agradecemos a nuestros compañeros y amigos que estuvieron con nosotros a lo largo de la carrera y que de una u otra forma nos ayudaron en diversas circunstancias.

A nuestro tutor Lic. Martín José Alonso por habernos animado a convertir una idea en lo que hoy es nuestro proyecto de graduación, a los profesores que estuvieron dispuestos a compartirnos sus conocimientos, en especial a quienes más que enseñarnos a leer y escribir nos enseñaron a estudiar y a los que nos enseñaron a aprender. A todos ellos, gracias.

"Confía en Jehová con todo tu corazón, y no te apoyes en tu propio entendimiento. En todos tus caminos tómalo en cuenta, y él mismo hará derechas tus sendas"
Proverbios 3:5,6

Contenido **Página**

I. INTRODUCCIÓN

La elaboración de la planilla de pagos o nómina surge de la necesidad de documentar los gastos en que incurre un ente económico o empresa en concepto de sueldos y salarios al personal que labora en la misma, de manera que se pueda disponer de la información tanto en detalle como totalizada, para la dirección de la empresa misma y para los entes reguladores de las relaciones laborales y de las actividades a las que se dedica la empresa.

En Nicaragua, es a partir del año 1982, cuando se aprueba la **"LEY CREADORA DE LA PLANILLA NACIONAL DE PAGO",** que se reglamenta jurídicamente el uso obligatorio para todo empleador establecido en el territorio nacional de la planilla de pagos, la que establece (la ley) los tipos en los que se deberá presentar la información de pago de salarios a los trabajadores; estos son:

PNP-1: Para el pago de salario fijo por semana, catorcena, quincena, mes y de 28 días.
PNP-2: Para el pago por destajo o por tarea, por hora y por día.

Se establece asimismo que la planilla de pagos tiene valor de prueba en materia fiscal y laboral, de modo que la falta de presentación de la planilla de pagos sin causa justificada en un conflicto laboral hará presumir, salvo prueba en contrario, como ciertos los datos aducidos por el trabajador respecto a las prestaciones reclamadas siempre y cuando este último demuestre la existencia de la relación laboral por cualquier medio legal de prueba.

Lo que debió ser una sencilla operación consistente en anotar el nombre, salario y cargo del trabajador se convierte, con el paso de los años, en una tarea un tanto complicada por la implementación de aspectos de carácter legal, como los mencionados anteriormente, en sus cálculos y presentación, más aún cuando se trata de una empresa con un considerable número de trabajadores.

El pago de horas extras, comisiones por ventas, adelantos de salarios, amortización de préstamos personales deducibles de planilla, son algunas operaciones que complican más la elaboración de una planilla de pagos.

Con la aparición de las hojas de cálculo como Microsoft Excel en la década de los años 90's el personal encargado de la elaboración de la nómina de las empresas que no tenían sistemas computarizados para la elaboración de la planilla de pagos cuenta con una poderosa herramienta que permite realizar los cálculos con mayor rapidez y precisión. Gracias a las funciones cada vez mejoradas que posee Microsoft Excel se pueden preparar de manera automática una planilla de pagos, ahorrando tiempo y esfuerzo.

Atendiendo la necesidad de minimizar el tiempo y personal utilizado en la elaboración de la nómina y aprovechado los avances tecnológicos en el área de la informática las grandes empresas han optado por desarrollar programas o aplicaciones en los distintos lenguajes de programación, que con el paso de los años han ido apareciendo, en los que se pueden registrar las operaciones diarias de las empresas para luego obtener de manera automatizada

los reportes requeridos, tanto por la administración como por los órganos reguladores de las actividades que realiza la empresa, incluyendo la planilla de pagos.

Sin embargo, las pequeñas y medianas empresas, como la mayoría en nuestro país, que no tienen la capacidad económica para costear la adquisición y mantenimiento de un sistema de gestión integral que le permita manejar sus operaciones continúan utilizando la hoja de cálculo de Microsoft Excel para la elaboración de su planilla de pagos.

En vista de lo anterior hemos decidido crear un programa en Microsoft Excel, utilizando el editor de Visual Basic para Aplicaciones, para la elaboración de la planilla de pagos de una empresa. El programa permite la manipulación de la información utilizada en los cálculos de manera que se pueda modificar o corregir en caso de errores, genera un reporte que muestra los totales en concepto de sueldos y salarios, deducciones y retenciones, pago de horas extras, comisiones y por supuesto la planilla de pagos.

A diferencia de la tradicional forma de utilizar Excel en nuestro programa no se realizan las operaciones en las celdas de la hoja de cálculo ya que éstas las hemos programado utilizando macros definidas en el editor de Visual Basic. El usuario entra los datos requeridos por el programa a través de los formularios que hemos diseñado, procesa dicha información y regresa los resultados directamente a la hoja de cálculo, la que posteriormente podrá ser modificada o utilizada para generar los reportes requeridos.

Con este programa eliminamos los inconvenientes que se producen al operar fórmulas en la hoja de cálculo cuando necesitamos borrar datos, eliminar o insertar celdas, filas o columnas o al cerrar el libro de trabajo sin guardar los cambios realizados.

Con esto queremos mostrar que Excel es un programa con un gran potencial, muchas veces subutilizado por estudiantes y profesionales; que con un adecuado uso de sus funciones y herramientas puede simplificarnos las tareas diarias y aumentar la eficiencia y productividad laboral.

II. OBJETIVOS

Objetivo General

Crear una aplicación en el programador de Microsoft Excel 2007 para la elaboración de la planilla de pagos de una empresa.

Objetivos Específicos

- Describir los elementos que integran la planilla de pagos y la forma de cálculo de las asignaciones y deducciones salariales de acuerdo a la normativa existente.
- Identificar las herramientas y funciones del programador de Microsoft Excel 2007 (Visual Basic para Aplicaciones) utilizadas en el diseño y creación del programa.
- Describir el algoritmo y componentes del programa.

III. PRINCIPALES ELEMENTOS DE LA PLANILLA DE PAGOS

1) Concepto

La planilla de pagos es una lista conformada por el conjunto de trabajadores a los cuales se les va a remunerar por los servicios que éstos le prestan al empleador, éste instrumento permite de una manera ordenada realizar el pago de sueldos o salarios a los trabajadores, además proporciona información contable y estadística, tanto para la empresa como para el ente encargado de regular las relaciones laborales.

2)Diseño de Nóminas

De acuerdo a la magnitud y las características de la empresa, se debe diseñar el modelo de la nómina apropiada (única, por departamento, sección o sucursal, etc.), el cual cambiará sustancialmente de un empresa a otra, sujeto a las variaciones de asignaciones, deducciones, acumulativos u otros determinados por la necesidad y naturaleza de las operaciones de la empresa.

3) Tipos de Nóminas

Básicamente, la nómina o planilla de pagos se clasifica por su presentación y forma de cálculo o según la periodicidad de pago.

Según su presentación:

- **Manual**

Se elabora por cálculos enteramente manuales, usadas en empresas pequeñas o de mediana capacidad.

- **Computarizada**

Se elabora a través de un computador, el cual permite, generalmente por medio de codificaciones, calcular automáticamente, tanto los aportes como los descuentos efectuados en el periodo a pagar; utilizado por empresas de considerable tamaño y que cuentan con sistemas de control de asistencia informatizados.

Según la forma de pago

- **Semanal**

Para obreros o personal de nómina diaria, cuyo lapso de pago esta establecido en ese tiempo.

- **Quincenal**

Este es el tipo de nómina más utilizado ya que aunque los contratos de trabajo estipulan salarios mensuales, generalmente se conviene que el pago de salarios se haga proporcional quincenalmente, por esta razón este tipo se usa para los empleados cuyo lapso de pago esta establecido en ese tiempo.

4) Normas de Control Interno para la Nómina

Las normas de control que se citarán a continuación, representan la garantía para la empresa, que los recursos monetarios destinados a la administración del personal, están adecuadamente distribuidos.

1. Deben ser aprobadas por personas autorizadas:
   a. La contratación de personal.
   b. El salario a pagar.
   c. Las deducciones que se realizan en nómina.
   d. Horas extraordinarias y otros pagos especiales.

2. El expediente de cada trabajador debe permanecer inaccesible para las personas que:
   a. Preparen la nómina.
   b. Aprueben nómina.
   c. Pagan la nómina.

3. Deben ser utilizados relojes, marcadores u otros medios que permitan controlar el tiempo de trabajo.

4. Debe existir separación de funciones:
   a. Entre el personal que controla el tiempo trabajado, elabora la nómina y paga la nómina.
   b. Entre el personal que contrata o despide y el que elabora la nómina.

5. Las contrataciones y despidos deben ser inmediatamente notificados al departamento de nómina.

6. El tiempo a pagar a cada trabajador debe ser aprobado por el jefe del departamento.

7. Los cálculos hechos en la nómina deben ser revisados antes de pagar a los trabajadores.

8. Cuando la nómina sea pagada con cheque:
   a. Debe existir una cuenta bancaria que se utilice sólo con ese objetivo.
   b. La cuenta bancaria de nómina debe ser conciliada mensualmente por una persona diferente de la que prepara la nómina y de la que paga.
   c. La persona que realiza la conciliación, debe recibir directamente el estado de cuenta bancario.

9. Cuando la nómina se paga en efectivo:
   a. Debe hacerse un retiro en el banco exactamente por el neto a pagar.
   b. Deben ser tomadas las medidas necesarias para evitar el robo por parte de empleados o extraños.

10. Cada trabajador, al recibir su remuneración, debe firmar un recibo en el que aparezcan detallados los diferentes conceptos que han intervenido en el cálculo.

11. El pago de la nómina debe ser periódicamente supervisado por un funcionario independiente del departamento de nómina.

12. Los sueldos no reclamados, deben ser custodiados por una persona diferente a la que elabora la nómina o hace los pagos, y debe enviar una relación de los mismos al departamento de contabilidad.

13. Los sueldos no reclamados, deben ser depositados en el banco después de un tiempo prudencial, registrando el correspondiente pasivo por ese concepto.

5) Partes de la nómina

En primer lugar, la planilla de pagos deberá proporcionar a modo de información los datos básicos de la empresa y el trabajador, esto es, nombre y/o razón social de la empresa, número de registro único de contribuyente, dirección, nombre y cargo del trabajador, el período para el cual se elabora, en segundo lugar se deberá detallar las asignaciones y deducciones salariales mediante las cuales se calcula el neto a recibir por el trabajador y finalmente se deberá identificar el personal responsable de la preparación y autorización de la nómina.

Ingresos	Deducciones Salariales
• Sueldo o salario • Horas extras • Vacaciones • Comisiones • Antigüedad • Otros	• Seguro social • Impuesto sobre la Renta • Aporte al sindicato • Préstamo Compañía • Otros

INGRESOS

- **Sueldo o Salario:**

El término sueldo se refiere al pago por servicios gerenciales, administrativos o similares. El término salario se refiere al pago por trabajo manual ya sea o no calificado. En la práctica se suele usar indistintamente los términos sueldo y salario para expresar la retribución que recibe un trabajador por la prestación de sus servicios a un empleador.

El salario constituye el centro de las relaciones de intercambio entre las personas y las empresas. Todas las personas dentro de las empresas ofrecen su tiempo y su fuerza y a cambio reciben dinero, lo cual representa el intercambio entre derechos y responsabilidades recíprocas entre el empleado y el empleador.

Así, el Salario es toda retribución que percibe el hombre a cambio de un servicio que ha prestado con su trabajo, es decir;

"La remuneración sea cual fuere su denominación o método de cálculo que pueda evaluarse en efectivo, debida por un empleador a un trabajador en virtud de los servicios u obras que este haya efectuado o debe efectuar, de acuerdo con lo estipulado en el Contrato de Trabajo."

Los sueldos y salarios pagados a los empleados son gastos laborales para el patrón y por tanto son deducibles a efectos fiscales.

El salario o sueldo base de un empleado puede incrementarse por comisiones, reparto de utilidades o ajustes inflacionarios. En muchas empresas se paga a los directivos o trabajadores un bono anual además de su salario base. El importe de este bono suele depender de alguna medición de la productividad, como los ingresos o la utilidad de la empresa.

- **Horas extras:**

El código del trabajo define las horas extraordinarias como el trabajo que se realiza fuera de la jornada ordinaria, séptimos días, feriados nacionales y asuetos decretados; no se consideran trabajo extraordinario el que se realice fuera de la jornada habitual para subsanar errores imputables al trabajador.

Las horas extraordinarias que labore el trabajador se pagarán un cien por ciento más de lo estipulado para la jornada normal respectiva. Las horas extraordinarias laborales no se pueden compensar con horas trabajadas en jornadas ordinarias, cuando éstas hayan sido disminuidas por disposición unilateral del empleador, en todo caso el número de horas extraordinarias laboradas por un trabajador no podrá ser superior a tres horas diarias ni nueve semanales excepto en caso de interés social o fuerza mayor como las siguientes:

1. Para prevenir o eliminar sin demora las consecuencias de catástrofes o accidentes que puedan perjudicar la producción o los servicios.

2. Para realizar labores urgente de reparación de maquinaria, equipo o inmuebles, cuando el mal estado de los mismos ponga en peligro la salud o la vida de los trabajadores o de la población.

3. Para realizar labores urgentes destinadas al restablecimiento de los servicios públicos o reparar la consecuencia de desastre que afecte dichos servicios.

4. Para realizar trabajo estacional o por temporada intensa, cuando sea imposible aumentar el número de trabajadores por razones técnicas, climatológicas o por escasez de fuerza de trabajo.

El trabajar horas extra es de común acuerdo entre el empleador y el empleado y generalmente deben ser autorizadas por un superior ya que estas se requieren particularmente sólo en situaciones excepcionales, por ejemplo si se deseara cumplir con un plan de producción.

Para calcular la remuneración correspondiente a horas extraordinarias se debe tener en cuenta que la jornada ordinaria es de ocho horas diarias, seis días a la semana o cuarenta y ocho horas semanales para la jornada diurna y de siete horas diarias, seis días a la semana o cuarenta y dos horas semanales para la jornada nocturna, se consideran remunerados los séptimos días y meses de treinta días, así que la forma de cálculo sería la siguiente:

$$\text{Pago por horas extras} = 2\times \#\text{ horas extras} \times \frac{\frac{salario\ nominal}{30\ dias}}{8\ horas}$$

- **Vacaciones:**

Según la legislación laboral en Nicaragua todo trabajador tiene derecho a disfrutar quince días de descanso continuo y remunerado en concepto de vacaciones, por cada seis meses de trabajo ininterrumpido al servicio de un mismo empleador.

En caso de trabajadores del estado y sus instituciones disfrutarán de vacaciones con goce de salario desde el sábado de ramos al domingo de pascua inclusive; del veinticuatro de Diciembre al primero de Enero inclusive; y de dieciséis días más durante el año.

En todo los casos, por interés del empleador o del trabajador o cuando se trate de las labores cíclicas del campo o de servicio que por su naturaleza no deban interrumpirse, la época de disfrute de las vacaciones podrá convenirse en fecha distinta a la que corresponde.

Cuando por alguna situación, y estando de común acuerdo empleado y empleador, no se tome el periodo de descanso por vacaciones, estas serán pagadas en base al último salario ordinario devengado por el trabajador, en caso de salarios variables se calculará en base al salario ordinario promedio de los últimos seis meses.

- **Comisiones:**

Las comisiones son las cantidades o importes que se le dan a un trabajador, por lo general estos son vendedores, representantes o cobradores; sobre lo que se vende o se cobra y corresponde a un porcentaje sobre el importe de la operación.

El objetivo de la comisión es incentivar el esfuerzo del trabajador ya que éste obtendrá mayores ingresos cuando mayores ingresos genere para la compañía.

La comisión suele consistir en un porcentaje fijo aplicado sobre el precio de venta o sobre el monto de la operación sobre el cual se paga la comisión pero, también puede establecerse un diferente baremo en virtud de la línea de productos, el canal de distribución, la categoría del cliente, etc. La razón es que las compañías suelen remunerar mejor las ventas con mayor rentabilidad, el volumen de ventas o cobranzas. De esta manera existen dos formas de establecer las comisiones:

1. **Porcentaje fijo:** se paga un porcentaje fijo sobre los ingresos generados por el trabajador. Por ejemplo: 3% por las ventas realizadas por el trabajador durante un periodo.

2. **Porcentaje a escala:** se paga un determinado porcentaje en base al nivel de ingresos generados por el trabajador.

Por ejemplo:

Un 3% por ventas menores o iguales a C$50,000 y 3.5% sobre las ventas superiores a los C$50,000.

En ocasiones el porcentaje de comisión se desglosa en función de la actividad de ventas realizadas. Por ejemplo, una comisión del 3% puede descomponerse en:

- 1% por apertura del cliente
- 1% por negociación de precios
- 1% por seguimiento comercial

La razón es que al pasar un cliente a cuenta clave, centralizar negociación de compras o introducirse en un grupo de compra, puede comenzar a gestionarse por una instancia superior dejando de ser visitado por el vendedor. En este caso, el vendedor seguirá cobrando el 1% por el apertura del cliente pero no el 2% adicional.

En el comercio minorista, también es común cobrar una comisión sobre las ventas realizadas con el objetivo de fomentar la correcta atención a los clientes. En estos casos la comisión suele ser compartida entre todos los dependientes para que colaboren en la gestión comercial y no se disputen las ventas en el establecimiento. Así, el importe de la comisión generado a final de mes se distribuye entre todo el personal de ventas.

DEDUCCIONES SALARIALES

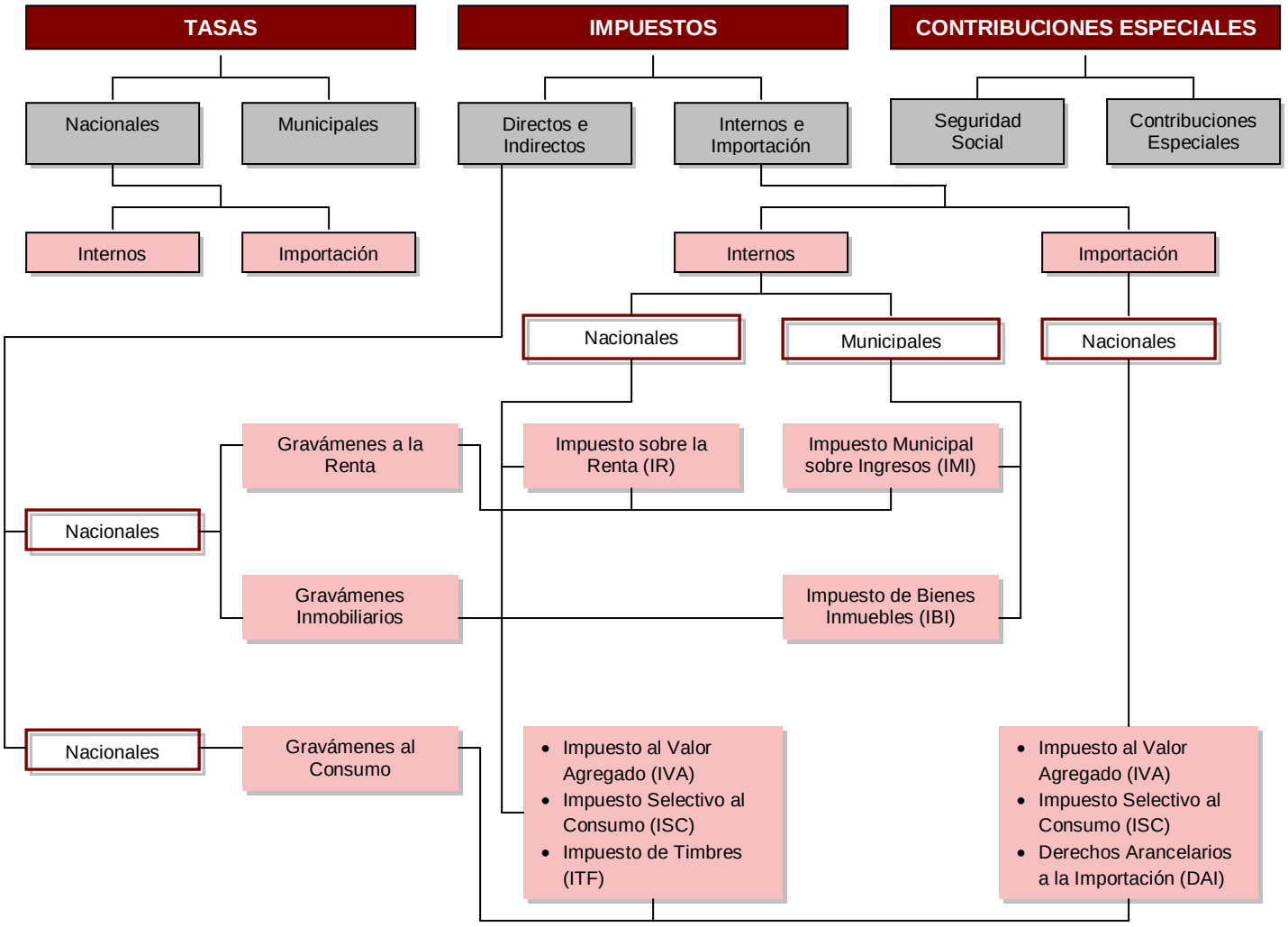
Las deducciones son todos los importes que se restan a los ingresos totales percibidos por el trabajador en el período de pago de la nómina para obtener el ingreso neto.

Las deducciones más importantes son las cuotas de aportaciones al seguro social y el impuesto sobre la renta, ya que éstas están reglamentadas según sus respectivas leyes.

Ni el patrón ni el empleado tienen alternativa sobre las deducciones de impuesto sobre la renta y el seguro social, pero sí los empleados pueden decidir que se le deduzcan cantidades adicionales para otros fines, por ejemplo pueden autorizar deducciones de ahorro para el retiro, contribuciones a organismos de beneficencia, primas de seguros para empleados, cuota sindical, cuotas de amortización de préstamos personales, etc.

A continuación se presenta un esquema de la Estructura Tributaria de Nicaragua, donde se puede apreciar el nivel que ocupan las deducciones mencionadas anteriormente dentro de la estructura tributaria nacional.

ESTRUCTURA TRIBUTARIA NACIONAL



- **Cuota al seguro social**

El Seguro Social es el instrumento del sistema de seguridad social establecido como servicio público de carácter nacional, cuyo objetivo es la protección de los trabajadores y sus familias frente a las contingencias sociales de la vida y del trabajo que de no ser así ocasionarían la desaparición o una fuerte reducción de los ingresos por causa de enfermedad, maternidad, riesgos profesionales, invalidez, vejez y muerte.

Por ello y basado en los principios de universalidad, integralidad, igualdad y solidaridad el seguro social otorga prestaciones de corto, mediano y largo plazo una vez que el asegurado cumple con los requisitos para recibirlas y de acuerdo al régimen en el que haya sido inscrito, entre éstas (prestaciones) podemos mencionar las pensiones por vejez, invalidez parcial o total, subsidio por maternidad, subsidio por enfermedad común o enfermedad profesional, etc.

En Nicaragua, según la ley de seguridad social, son sujeto de aseguramiento obligatorio todas las personas que desempeñan un trabajo o servicio profesional en calidad de dependiente en forma eventual, temporal o permanente a un empleador; sea persona natural o jurídica, independiente de la relación que los vincule y de la naturaleza de la actividad económica, éstos deberán ser inscritos en el seguro social.

El empleador esta obligado a inscribirse e inscribir a sus trabajadores en el Instituto Nicaragüense de Seguridad Social, que es el órgano encargado de aplicar, administrar, implementar y evaluar el cumplimiento de las normativas relativas a la seguridad social, así como comunicar los cambios en su personal y en las remuneraciones dentro de los plazos y términos establecidos.

- **Forma de cálculo:**

El sistema de cálculo para la deducción de las cotizaciones consiste en aplicar directamente el porcentaje que corresponde según el régimen de seguridad social al que se encuentra adscrito, a la remuneración total del trabajador.

El salario mínimo de aplicación es el salario mínimo vigente, el salario máximo sujeto de cotización es de C$ 37,518; los porcentajes de aplicación se harán de conformidad al régimen de aseguramiento, los cuales son:

1. Para los afiliados al régimen IVM y riesgos profesionales conforme a la siguiente tabla:

	IVM	Riesgos profesionales	Víctimas de guerras	Total
Empleador	7.00 %	1.50 %	1.5 %	**10.00 %**
Trabajador	4.00 %	0.00 %	0.25 %	**4.25 %**
Total	**11.00 %**	**1.50 %**	**1.75 %**	**14.25 %**

2. Para los afiliados al régimen integral conforme a la siguiente tabla:

	IVM	Riegos profesionales	Enfermedad y maternidad	Víctimas de guerras	Total
Empleador	7.00 %	1.50 %	6.00 %	1.50 %	**16.00 %**
Trabajador	4.00 %	0.00 %	2.25 %	0.00 %	**6.25 %**
Estado	0.00 %	0.00 %	0.25 %	0.00 %	**0.25 %**
Total	**11.00 %**	**1.50 %**	**8.50 %**	**1.50 %**	**22.50 %**

- **El Impuesto sobre la Renta**

El impuesto sobre la renta (IR), creado en el año 1974 por el decreto No. 662 y vigente a partir del primero de Enero de 1975, es el impuesto que se aplica a toda renta neta originada en Nicaragua, obtenida por personas jurídicas y naturales, cualquiera que sea su nacionalidad, domicilio o residencia, inclusive si el bien o servicio se paga fuera de Nicaragua.

A efectos de obtener la renta neta grabable con el IR se deducirá a la renta bruta los gastos pagados y causados necesarios para la existencia o mantenimiento de la fuente generadora de renta gravable, tales como intereses por deudas a cargo del contribuyente, costo de bienes o mercancías vendidas, primas de seguros, depreciación de bienes depreciables, la cotización de todas las personas naturales asalariadas en cualquiera de los regímenes de la seguridad social, para las empresas de seguros los incrementos en las reservas técnicas y matemáticas y las que se dispongan a prevenir devoluciones de pólizas no devengadas, dichas reservas están determinas por las normativas de la superintendencia de bancos y otras instituciones financieras, etc.

La renta bruta comprende todos los ingresos recibidos y los devengados por el contribuyente durante el año gravable, en forma periódica, eventual u ocasional, sean éstos en dinero efectivo, bienes y compensaciones provenientes de ventas, rentas o utilidades, originados por la exportación de bienes producidos, manufacturados, tratados o comprados en el país, prestación de servicios, arriendos, subarriendos, trabajos, salarios y demás pagos que se hagan por razón del cargo, actividades remuneradas de cualquier índole, ganancias o beneficios producidos por bienes muebles o inmuebles, ganancias de capital y los demás ingresos de cualquier naturaleza que provengan de causas que no estén exentas en la ley.

Particularmente en este trabajo nos ocupa el estudio y forma de cálculo del impuesto sobre la renta de las personas naturales asalariadas por lo que nos centraremos en ese sentido dejando para otros trabajos el estudio a profundidad del impuesto sobre la renta para las personas naturales no asalariadas y personas jurídicas.

Obviamente que la cantidad que se debe retener por impuesto sobre la renta difiere según el sueldo base y otros ingresos del empleado de acuerdo a la tabla de retención vigente a la fecha.

Actualmente el impuesto sobre la renta es reglamentado por la ley de equidad fiscal, reformada recientemente, de modo que la forma de cálculo expuesta en este trabajo esta de acorde a las reformas puestas en vigencia a partir de Enero de 2010.

Ingresos no gravables con el IR

No serán sujeto de retención de IR los ingresos que reciban los trabajadores o sus familiares en concepto de indemnizaciones contempladas en el código del trabajo, convenios colectivos, y los de cualquier otra índole laboral o ingresos que se perciban de acuerdo a la legislación de seguridad social así como el aguinaldo o treceavo mes.

Forma de cálculo

Anteriormente mencionamos que la ley de equidad fiscal ha sido reformada y que los cálculos los haremos de acuerdo a esas modificaciones.

Aclaramos que, de acuerdo a las reformas de la ley de equidad fiscal para las personas naturales el pago del IR se continúa haciendo igual, esto es de acuerdo a la siguiente tabla de tarifas progresivas:

Renta gravable (Estratos)		Impuesto Base	Porcentaje Aplicable	Sobre exceso de
De C$	Hasta C$			
1.00	50,000.00	0.00	0%	0.00
50,000.01	100,000.00	0.00	10%	50,000.00
100,000.01	200,000.00	5,000.00	15%	100,000.00
200,000.01	300,000.00	20,000.00	20%	200,000.00
300,000.01	500,000.00	40,000.00	25%	300,000.00
500,000.01	a más	90,000.00	30%	500,000.00

En cambio para las personas asalariadas en particular, el impuesto a pagar se calculará de acuerdo a la siguiente tabla de tarifas progresivas:

Renta gravable (Estratos)		Impuesto Base	Porcentaje Aplicable	Sobre exceso de
De C$	Hasta C$			
1.00	75,000.00	0.00	0%	0.00
75,000.01	100,000.00	0.00	10%	75,000.00
100,000.01	200,000.00	2,500.00	15%	100,000.00
200,000.01	300,000.00	17,500.00	20%	200,000.00
300,000.01	500,000.00	37,500.00	25%	300,000.00
500,000.01	a más	87,500.00	30%	500,000.00

De acuerdo a esto el cálculo del impuesto sobre la renta se hará de la siguiente manera:

I. Personas con ingresos mensuales (deducida la contribución al seguro social) menor a C$6,250.00 no pagan IR.

II. Para las personas con ingresos mensuales superiores a C$6,250.00
   a. Deducir a los ingresos del mes la cuota de aportación al seguro social.
   b. Obtener los ingresos anuales esperados de acuerdo a los ingresos del mes.
   c. Aplicar el porcentaje correspondiente sobre el exceso, según el estrato en el que se encuentren dichos ingresos.
   d. Sumar el impuesto base al resultado obtenido en "c" para obtener el impuesto anual y luego calcular proporcionalmente el impuesto correspondiente al periodo en el que se obtuvieron los ingresos.

Declaración de la renta

Está obligado a hacer declaración de renta ante la dirección general de ingresos toda persona natural cuya renta bruta exceda los cincuenta mil córdobas durante todo el año gravable y toda persona jurídica cualquiera que sea la cuantía de su renta, aún cuando ésta este exenta de impuesto. Para las personas naturales asalariadas cuyos únicos ingresos consistan en salarios o remuneraciones de una sola fuente, no están obligados a presentar declaración de renta, excepto los que hayan contratado invertir su renta según la ley de incentivos turístico de la República de Nicaragua.

En el caso de una persona naturales asalariadas que obtenga ingresos de diferentes fuentes, que consolidado excedan los setenta y cinco mil córdobas anuales estarán obligados a declarar y pagar el correspondiente impuesto sobre la renta.

IV. PROGRAMACIÓN EN VISUAL BASIC PARA APLICACIONES

1) Macros en Microsoft Excel

Una *Macro* es un programa escrito o grabado por el usuario que almacena una serie de comandos de Microsoft Excel que pueden utilizarse posteriormente como un único comando. Mediante las macros pueden automatizarse las tareas complejas y pueden reducirse el número de pasos necesarios para realizar las tareas que se ejecutan con más frecuencia. Las macros se graban en el lenguaje de programación de Visual Basic para Aplicaciones de Microsoft. También pueden escribirse las macros directamente en el Editor de Visual Basic para Aplicaciones.

- **Como grabar un macro**

La forma más fácil e intuitiva de crear macros es hacerlo mediante la grabadora de macros del que dispone Excel.

La grabadora de macros permite grabar las acciones deseadas que posteriormente las traduce a instrucciones en VBA (*Visual Basic for Applications*), las cuales podemos modificar posteriormente si tenemos conocimientos de programación.

Antes de grabar o escribir una macro hay que tener en cuenta los siguientes puntos:

1. Planifique los pasos y los comandos que desea que ejecute la macro.

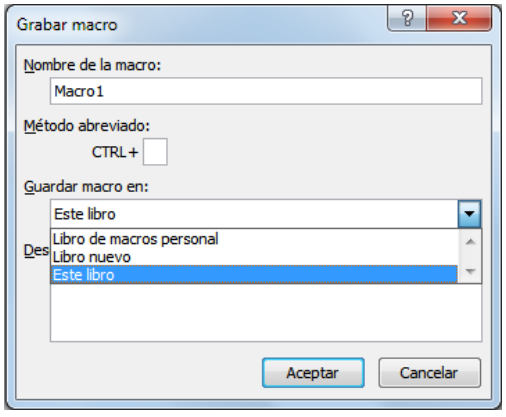
2. Si se comete algún error mientras se graba la macro, también se grabarán las correcciones que se realicen.

3. Cada vez que se grabe una macro, ésta se almacenará en un nuevo módulo adjunto a un libro.

Para grabar un macro se tiene que hacer lo siguiente:

Active la grabadora de macros haciendo clic en la ficha programador y luego en **Grabar macro**.

Aparece la ventana siguiente.



En **Nombre de Macro**, ponga *el nombre deseado*. En **Guardar Macro en**, deje la opción **Este libro**, la opción **Método abreviado** permite ejecutar la macro presionando la tecla CTRL + la tecla que ponga en el recuadro. Se debe escoger una tecla que no tenga un acceso directo predeterminado para Windows.

Pulse sobre el botón Aceptar y la macro estará lista para operar.

Grabaremos una macro y veremos sus códigos:

Para observar los códigos de una macro debemos de seguir los siguientes pasos:

1. Primeramente trasládese a la celda ***A5*** antes de empezar la grabación de la Macro.

2. Presione el Botón **Grabar Macro** de la ficha programador**. Excel** muestra el cuadro de Dialogo Grabar Macro.

3. En la opción **Método Abreviado** escriba la letra **r**, por lo tanto la macro se llamará con **Control + r.**

4. Presione el botón **Aceptar. Excel** inicia la grabación del la **Macro1.**

5. Trasládese a la celda **A1** y escriba ***"Su Nombre"***, después presione **Enter** para aceptar el valor en la celda.

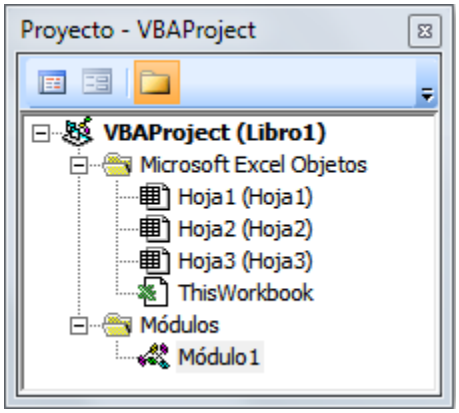
6. Pare la grabación de la macro presionando el botón **Detener Grabación** de la ficha programador**.** Excel a grabado los pasos y a generado un código, observémoslos:

7. Presione la tecla Alt + la tecla de función F11 (**Alt + F11**). Excel nos traslada al Editor de Visual Basic. Si este editor no se activa es que Excel no esta bien instalado o se ha borrado. También puede acceder desde Excel haciendo clic en la ficha **Programador** y luego en la opción Visual Basic.

8. Active los siguientes cuadros o ventanas:

    - De clic en el **Menú Ver** y elija la opción **Explorador de Proyectos**
    - De clic en el **Menú ver** y elija la opción **Ventana Propiedades**

Estas dos opciones deben de estar siempre activadas ya que de ahí depende todo lo que vayamos a hacer.



9. En el cuadro **Proyecto** de doble clic en **Módulos** o simplemente presione el signo de + que aparece en la opción **Módulos**. Se activara debajo de **Módulos** la Opción **Modulo1**

10. De doble clic en **Modulo1**. Se mostrara en el Editor de Visual Basic el código de la macro que grabamos de la siguiente forma:

```
Sub Macro1()
'
' Macro1 Macro
'
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Su Nombre"
    Range("A2").Select
End Sub
```

Que es lo que significa esto, a continuación se da una explicación de lo que ha hecho Excel.

- **Sub** y **End Sub** indican el inicio y el final del procedimiento de la **Macro1**

- Todo lo que aparece con un apóstrofe indica que no se tomará en cuenta, que es sólo texto o comentarios y ese texto debe de aparecer en un color, generalmente color verde.

- **Range("A1").Select** Indica que lo primero que hicimos al grabar la macro fue trasladarnos a la celda **A1**. La orden **Range** nos permite trasladarnos a una celda

- **ActiveCell.FormulaR1C1 = "*"Su Nombre"*"** Esto indica que se escribirá en la celda en que se encuentra el valor de texto ***"Su Nombre"***. Todo lo que aparece entre comillas siempre será un valor de texto. La orden **ActiveCell.FormulaR1C1** nos permite escribir un valor en la celda activa.

- **Range("A2").Select** Otra vez indicamos que se traslade a la celda A2. Esto se debe a que cuando escribimos el nombre de ***"Su Nombre"*** en **A1** presionamos **Enter** y al dar **Enter** bajó a la celda **A2**.

Para comprender mejor alteraremos el código dentro del editor de Visual Basic.

Que crees que pasara aquí con nuestra Macro:

```
Sub Macro1()
'
' Macro1 Macro
'
' Acceso directo: CTRL+r
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Su Nombre"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "Barrio La Primavera"
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "12/23/1970"
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "Managua, Nicaragua"
    Range("A5").Select
End Sub
```

Así es acabado de alterar el código y cuando regrese a **Excel** y ejecute la macro con **Control + r** hará lo siguiente:

En A1 escribirá *"Su Nombre"*
En A2 escribirá "Barrio la primavera"
En A3 escribirá "23/12/1970"
En A4 escribirá "Managua, Nicaragua"

Así que salimos del editor dando clic en el **Menú Archivo** y eligiendo la opción **Cerrar y volver a Microsoft Excel**. Si no desea salir por completo de clic en **el botón Microsoft Excel** que se encuentra activado en la barra de tareas y cuando deseé volver al editor de clic en el **botón Microsoft Visual Basic** que se encuentra en la barra de tareas.

Ahora que ya salimos de **Visual Basic** y estamos en **Excel** de nuevo ejecutemos la macro presionando **Control + r** y veamos los resultados de nuestra modificación.

2) Conceptos básicos de programación en VBA

Queremos aclarar que aunque nuestro principal objetivo es crear un programa para la elaboración de la planilla de pagos consideramos que, sin llegar a convertir este trabajo en un manual para programación en Excel, es necesario explicar al menos los aspectos

relevantes de la programación y las estructuras utilizadas en el diseño y creación de nuestro programa por tratarse de un tema relativamente desconocido.

- **Objetos propiedades y métodos.**

Al trabajar con macros en Excel deben tenerse claros algunos conceptos de lo que se llama Programación Orientada a Objetos (OOP), o Programación Basada en Objetos que es más bien el tipo de programación utilizado por VBA.

- ***Objetos***

Cuando nos referimos a objetos hablamos de algo más o menos abstracto que puede ser cualquier cosa, una casa, un carro etc.

En OOP la generalización de un objeto se llama clase, así la clase carro representa todos los carros del mundo, es decir, define como es un carro; algo que tiene cuatro ruedas, un chasis y un motor es un objeto de la clase carro.

- ***Propiedades***

Las propiedades son las características modificables o no de un objeto la que se definen en la clase y luego se particularizan en cada objeto, por ejemplo, en la clase carro se pueden definir las propiedades color, ancho y largo, luego al definir un objeto concreto estas propiedades se particularizan, por ejemplo Color = Rojo, Ancho = 1.8 metros, Largo = 3 metros.

- **Métodos**

La mayoría de los objetos tienen comportamientos o realizan acciones, en nuestro ejemplo el objeto carro puede moverse. Cualquier proceso que implique una acción o pauta de comportamiento por parte de un objeto se define en su clase para que pueda manifestar en cualquiera de sus objetos. De modo que en la clase carro se definen todos los procesos necesarios para llevarlo a cabo y luego cada objeto de la clase carro sólo tendrá que invocar este método para moverse.

Visto todos estos conceptos desde Excel nos encontramos con objetos como WorkSheets (objeto hoja de calculo) o Range (objeto celda o rango de celda), así el objeto Range esta definido por una clase donde se define sus propiedades, por ejemplo: Value, valor de la celda; Row, que contiene la fila de la celda; Font, que contiene la fuente de los caracteres que muestra la celda.

Range como objeto tiene métodos, por ejemplo, el método Activate activa una determinada celda. Es habitual que una propiedad de un objeto sea un objeto por ejemplo: en Excel el objeto WorkSheets tiene la propiedad Range que es un objeto, Range tiene la propiedad Font que también es un objeto y finalmente Font tiene la propiedad Bold. Excel es un

objeto llamado Application y contiene otros objetos como WorkBook y CommandBar, el objeto WoorkBook puede contener otros objetos como WorkSheet y Chart. Un objeto WoorkSheet puede contener objetos como Range, PivotTable y demás.

Excel proporciona más de 100 clases de objetos para manipular.
Ejemplos de objetos son un libro de trabajo, una hoja de cálculo, un rango de una hoja de cálculo, un gráfico y un rectángulo dibujado. Existen muchos más objetos a nuestra disposición y se pueden manipular mediante el uso de código VBA.

Objetos similares forman una colección. Por ejemplo, la colección WorkSheets está compuesta por todas las hojas de cálculo de un libro concreto. La colección CommandBars está compuesta por todos los objetos CommandBar. Las colecciones son objetos en sí mismas. Cuando nos referimos a un objeto contenido o miembro, estamos especificando su posición en la jerarquía del objeto, usando un punto como separador entre el contenedor y el miembro.

- **Procedimientos y funciones.**

Se define como procedimiento o función a un bloque de código que realiza una tarea especifica. Algunos problemas se pueden resolver utilizando un único procedimiento, pero a medida que el programa crece se hace necesario la introducción de más líneas de código, lo que complica la programación, ya que grandes bloques de códigos implican mayor complicación, repetición de sentencias y lo que es peor, mayores problemas a la hora de depurar errores, ampliar funcionalidades o hacer modificaciones.

Utilizando procedimientos podemos tratar cada problema o tarea de forma más o menos aislada de forma que construiremos el programa paso a paso sin tener que resolver o controlar múltiples cosas a la vez, así que los diferentes pasos que se deban ejecutar para que un programa haga algo quedarán bien definidos es su correspondiente procedimiento, si ocurre un error lo podemos localizar rápidamente en su respectivo procedimiento.

- **Forma de definir un procedimiento**

En general un procedimiento se define de la siguiente manera:

```
Sub Nombre_procedimiento()
      Sentencias
End Sub
```

Es decir, un procedimiento tiene un inicio y un fin definido por la "orden" *Sub* para iniciar y *End Sub* para finalizar, un nombre para que Visual Basic lo reconozca y lo podamos llamar desde otro procedimiento y un bloque de sentencias que cuando se ejecuten devuelvan el resultado deseado.

- **Variables**

No es indispensable definir variables a la hora de programar en Excel ya que al final la hoja de cálculo es donde se guardará el texto que escribamos o el resultado de operaciones programadas en el bloque de código.

Sin embargo, trabajar con variables es mucho más cómodo y le brinda elegancia al programa. En nuestro caso hemos optado por trabajar con variables ya que así no tendremos que estar recordando celdas o rangos de celdas sino, nombres de variables relacionadas con el tema.

Para Visual Basic una variable es simplemente un trozo de memoria que la función o procedimiento se reserva para guardar datos de manera temporal, esto es, el valor de la variable será el que se le asigne cada vez que se ejecute una sentencia o bloque de códigos (procedimiento).

La forma general de declarar una variable es la siguiente:

```
Dim Nombre_variable As Tipo
```

Ejemplo:

```
Dim Edad As Integer
```

Con esto indicamos al procedimiento o función que se reserve un trozo de memoria que se llama *Edad* y que el tipo de datos que se guardarán ahí son números enteros.

Para aclarar un poco, haremos un programa que pida la edad de una persona y la guarde en la celda "B2" de la hoja activa usando variables y luego sin usar variables.

```
Sub Entrar_edad()
    Dim Edad As Integer
      Edad = InputBox ("Introduzca la edad", "Entrar edad")
      ActiveSheets.Range("B2").Value = Edad
End Sub
```

Luego sin variables:

```
Sub Entrar_edad()
    ActiveSheets.Range("B2").Value = InputBox("Introduzca la edad",
    "Entrar edad")
End Sub
```

Separando la función *InputBox*, que luego explicaremos, consideramos que hay mayor claridad en el código del procedimiento donde definimos la variable edad; ya que primero se pregunta la edad, se guarda en la variable edad y luego indicamos que la edad se guarde en la celda *"B2"* de la hoja activa mientras que sin usar variables debemos realizar toda la

tarea en una misma línea de código. En este caso se trata de algo sencillo, trate de imaginar que lo que queremos en determinada celda es el producto de una serie de operaciones matemáticas que involucre más de dos valores y comprenderá que es mucho mejor usar variables.

Tipos de datos en visual Basic para Excel

La siguiente tabla muestra los tipos de datos que podemos asignar a una variable definida en Visual Basic for Applications con una breve descripción del tamaño de almacenamiento y valores admitidos.

Tipo de datos	Tamaño de almacenamiento	Intervalo
Byte	1 byte	0 a 255
Bolean	2 bytes	**True** o **False**
Integer	2 bytes	-32,768 a 32,767
Long (entero largo)	4 bytes	-2,147,483,648 a 2,147,483,647
Single (coma flotante/ precisión simple)	4 bytes	-3,402823E38 a –1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
Double (coma flotante/ precisión doble)	8 bytes	-1.79769313486231E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
Currency (entero a escala)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
Decimal	14 bytes	+/-79.228.162.514.264.337.593.543.950.335 sin punto decimal; +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/-0,0000000000000000000000000001
Date	8 bytes	1 de enero de 100 a 31 de diciembre de 9999
Object	4 bytes	Cualquier referencia a tipo **Object**
String (longitud variable)	10 bytes + longitud de la cadena	Desde 0 a 2.000 millones
String (longitud fija)	Longitud de la cadena	Desde 1 a 65.400 aproximadamente
Variant (con números)	16 bytes	Cualquier valor numérico hasta el intervalo de un tipo **Double**
Variant (con caracteres)	22 bytes + longitud de la cadena	El mismo intervalo que para un tipo **String** de longitud variable
Definido por el usuario (utilizando **Type**)	Número requerido por los elementos	El intervalo de cada elemento es el mismo que el intervalo de su tipo de datos.

3) Estructuras de programación

- **Estructuras condicionales**

Las estructuras condicionales son instrucciones de programación que permiten controlar la ejecución de un fragmento de código en función de si se cumple o no una condición.

Las estructuras condicionales tienen dos formas:

```
I.   If Condición Then
      Sentencia 1
      Sentencia 2
               .
               .
               .
      Sentencia N

     End If
```

Cuando el programa llega a la instrucción *If condición Then* se evalúa la condición y si ésta se cumple se ejecutan todas las sentencias encerradas en el bloque, de lo contrario se salta esta sentencia.

```
II.   If Condición Then
       Sentencia 1
       Sentencia 2
                .
                .
                .
       Sentencia N
      Else
       Sentencia 1
       Sentencia 2
                .
                .
                .
       Sentencia N
      End If
```

Aquí se ejecuta un bloque de sentencia si se cumple la condición (primer bloque) y otra si no se cumple, esto se usa cuando se requiere una respuesta alternativa, si - si no. Lo más importante de las estructuras condicionales es que dentro de una estructura *If* puede ir otra, y de dentro de esta ir otra y otra...Así podemos mandar a ejecutar una sentencia o bloque de sentencias si se cumple más de una condición, entonces la estructura quedaría más o menos así:

```
If Condición 1 Then
     Sentencia 1
     Sentencia 2
         .
         .
         .
     Sentencia N
Else
   If Condición 2 Then
        Sentencia1
        Sentencia2
            .
            .
            .
        Sentencia N
   Else
     If Condición i Then
          Sentencia1
          Sentencia2
               .
               .
               .
          Sentencia N
               .
               .
               .
   Else
              .
              .
              .
     If Condiciones Then
        Sentencia 1
        Sentencia 2
            .
            .
            .
       Sentencia N

                End If
                   .
                   .
                   .
   End If
   .
   .
   .
End If
End If
```

Obsérvese que la segunda estructura *If…Else…End If* queda dentro del *Else* de la primera estructura, la tercera dentro de la segunda y así sucesivamente. Es una regla general, cuando se usa el *End If*, esta cierra siempre el último *If* (o *Else*) abierto.

- **Estructuras Repetitivas For (para)**

Esta estructura sirve para repetir la ejecución de una sentencia o bloque de sentencias un número definido de veces.

La estructura general en Visual es la siguiente:

```
For Variable = Valor_inicial to Valor_final Step Incremento
                          Sentencia 1
                          Sentencia 2
                                .
                                .
                                .
                          Sentencia N
Next Variable
```

Si el incremento es 1, no es necesario poner *step 1*

- **Recorrer Celdas de una hoja de cálculo.**

Una operación bastante habitual, cuando se trabaja con Excel es el recorrido de rango de celda para llenarlas con valores, mirar su contenido, aplicar formato etc. esta operación la podemos realizar utilizando las propiedades *Cell* y *Offset* y potenciarlas con el uso de estructuras repetitivas ("*For*", por ejemplo).

Propiedad Cells

Esta propiedad sirve para referenciar una celda o un rango de celda según coordenadas de filas y columnas ya que Excel asigna un número a cada fila y columna de la hoja de cálculo de modo que cada una de las celdas tiene una referencia única formada por su número de fila y columna correspondiente de la siguiente manera (fila, columna).

Por ejemplo, para referirnos a la celda "B3" de la hoja 1 escribimos *WorkSheets.Cell (3,2)* es decir, fila 3, columna 2. Además, esta propiedad la podemos potenciar con la estructura repetitiva *For* de la siguiente manera:

Por ejemplo, si necesitamos recorrer el rango de celda "A1:E5" de la hoja 1 para llenarlo con algún valor lo podemos hacer así:

```
Sub Ejemplo()

Dim i As Integer
Dim j As Integer

'Activamos la hoja 1

Hoja1.Activate
```

```
For i = 1 to 5
       For j = 1 to 5

              ActiveSheet.Cells(i, j).Value = "Algún valor "

         Next j
   Next i

End Sub
```

Como se ha observado, al igual que las estructuras condicionales, las repetitivas se pueden anidar una dentro de otra, en este caso ambas inician en 1 para referirnos a la celda "A1" luego incrementamos *j* (*Next j*) para pasar a la siguiente columna hasta llegar a cinco (columna E) el programa sale del bucle y pasa a incrementar *i* para ir a la fila dos e inicializa *j = 1* (celda A2). Cuando utilizamos *Cells* como propiedad del objeto *WorkSheets* (este caso) *Cells* cuenta a partir de la celda referenciada por *Range*. De modo que *ActiveSheetes.Cells (3,4)* se refiera a la celda "D3" y *ActiveSheets.Range("B2").Cells(3,4)* se refiere a la celda "E4".

Propiedad *Offset*

La propiedad *Offset* es una propiedad del objeto Range que sirve, al igual que la propiedad *Cells*, para recorrer un rango de celdas a través de una hoja de cálculo con la diferencia que *Offset* nos permite referenciar una celda situada a ±*n* filas y ±*m* columnas de una celda dada, por ejemplo, para referirnos a la celda "C3" estando en la celda "A1" lo hacemos de la siguiente manera:

```
ActiveSheet.Range("A1").Offset(2, 2).Value = "Algún Valor"
```

La celda "C3" dirá *"Algún Valor"*, la cual está ubicada a 2 filas y 2 columnas desde "A1".

```
ActiveCell.Offset(5,0).Value = "Algún Valor"
```

Este código asignará el texto "Algún Valor" a la celda que se encuentra a 5 filas por debajo de la celda activa.

```
ActiveCell.Offset(-3,2).Active
```

Activa la celda que está 3 filas por encima y dos columnas a la derecha de la celda activa.

De manera similar a la propiedad *Cells*, *Offset* se puede potenciar con el uso de estructuras repetitivas.

- **Estructura repetitiva Do While… Loop**

Como habrá observado, la estructura repetitiva *For* sólo se puede utilizar cuando se sabe previamente el número de veces que se ha de repetir un proceso, pero ¿Qué hacer cuando no se sabe con anticipación cuantas veces se ha de repetir un determinado código? En este caso debemos recurrir a la estructura repetitiva *Do While…Loop*, en alguna de sus formas. Esta estructura está controlada por una o varias condiciones, la repetición del bloque de sentencias dependerá de si se van cumpliendo las condiciones.

Su estructura básica es:

```
Do While (Se cumpla la condición)
     Sentencia 1
     Sentencia 2
          .
          .
          .
     Sentencia N
Loop
```

En esta forma el bloque de sentencias se ejecuta si se cumple la condición. Hay casos en los que se tienen que entrar datos desde el teclado, luego se procesa la información y devuelve un resultado, los programadores o usuarios quizá prefieran que luego de la ejecución el programa pregunte si se desea ingresar más datos, de manera que si el usuario contesta que sí, el programa se vuelve a ejecutar, si contesta que no, se finaliza el proceso. En este caso sería necesario escribir la siguiente línea antes del bucle.

```
Mas_datos = vbyes
```

De forma que el bucle completo quedaría de la siguiente manera:

```
Mas_datos = vbyes
Do While Mas_datos = vbyes
 Sentencia 1
 Sentencia 2
      .
      .
      .
 Sentencia N
Mas_datos = MsgBox ("Otros datos", vbyes + vbQuestion, "Entrada de
datos")
Loop
```

La línea anterior al bucle es necesaria para que cuando se evalúe la condición por primera vez ésta se cumpla y el usuario pueda ingresar los datos y el programa realice las tareas especificadas en la o las sentencias. Más adelante explicaremos la función *MsgBox*.

- **Estructura Do… Loop While**

Esta estructura es similar a la anterior con la diferencia que la condición se evalúa al final, por lo que las instrucciones del cuerpo del bucle se ejecutarán al menos una vez, lo que es ideal para la situación presentada anteriormente ya que si se van a entrar datos, entraremos al menos uno.

En Visual Basic la estructura general es la siguiente:

```
Do
 Sentencia(s)
Loop While (Se cumpla la Condición)
```

Así, si hacemos un programa para entrar datos desde el teclado y guardarlos en algún lugar de la hoja de cálculo, lo haríamos más o menos de la siguiente manera. (Vamos a pedir el nombre y la edad de una persona y lo guardaremos en la hoja1 a partir de la celda A2)

```
Sub Ejemplo()

'Definimos la variable Nombre como tipo texto para almacenar el nombre
  Dim Nombre As String

'Definimos la variable Edad como tipo entero para almacenar la edad
  Dim Edad As Integer

'Definimos la variable Mas_datos como tipo texto
  Dim Mas_datos As String

'Activamos la hoja 1
  Hoja1.Activate

'Activar la celda A2 de la hoja 1 (hoja activa)
  ActiveSheet.Range("A2").Activate

'Si la celda activa tiene algún valor activamos la siguiente (hacia
abajo)

  Do While Not IsEmpty(ActiveCell)
    Activecell.Offset(1,0).Activate
  Loop

  Do
    Nombre = InputBox("Entrar nombre", "Nombre")
    Edad = InputBox("Entrar edad", "Edad")

    With ActiveCell
         .Value = Nombre
         .Offset(0,1).Value = Edad
    End With
```

```
        Activecell.Offset(1,0).Activate
        Mas_datos = MsgBox("Otro registro?", vbYesNo + vbQuestion,
        "Entrada de datos")

      Loop While Mas_datos = vbYes

End Sub
```

Otra forma de éste tipo de estructuras es la siguiente:

```
Do
   Sentencias
Loop Until (se cumpla la condición)
```

Esta estructura evalúa la condición al final, igual que la anterior, con la diferencia que el bucle se ejecuta hasta que se cumpla la condición y no mientras se cumpla, así que en el programa anterior sólo tendríamos que cambiar la línea:

```
      Loop While Mas_datos = vbYes
```

Por:

```
      Loop Until Mas_datos = vbNo
```

Como se puede apreciar, es prácticamente lo mismo, (*hacer mientras sí* o *hacer hasta que no*), así que cuál de las tres formas utilizar (*Do While...Loop, Do...Loop While* o *Do...Loop Until*) dependerá del gusto del programador.

- **Instrucción With**

La sentencia *With* ayuda a tener que escribir menos código sin que por eso el programa pierda claridad, concretamente esta función sirve para ejecutar una serie de acciones sobre un mismo objeto. Su sintaxis es la siguiente:

```
With Objeto
   Instrucciones
End With
```

Como recordará, esta función la utilizamos en el programa de ejemplo anterior para guardar los datos de entrada digitados desde el teclado, sin ella el código habría sido de la siguiente manera:

```
ActiveCell.Value = Nombre
ActiveCell.Offset(0,1).Value = Edad
```

La ventaja de utilizar la instrucción *With* es que nos ahorramos código al no tener que ir referenciando la celda en cada línea de código.

- **Estructura Select Case**

En ocasiones se dará el caso de que se ejecutará una sentencia o bloque de sentencias en función del valor que tenga o pueda tener una variable, una celda o una expresión, en estos casos se pueden usar las estructuras condicionales mostradas anteriormente.

Aunque en ocasiones resulte factible el uso de este tipo de estructuras para solucionar estos problemas por cuestión de estética y legibilidad del programa es preferible utilizar la estructura *Select Case* la que como se mencionó anteriormente permite ejecutar una sentencia o bloque de sentencias en dependencia del valor de una variable, celda, etc.

La sintaxis de la estructura *Select Case* en VBA es la siguiente:

```
Select Case Expresión_prueba

  Case Valores:
    Instrucciones

  Case Valores:
    Instrucciones
        .
        .
        .

  Case Valores:
    Instrucciones

  Case Else:
    Instrucciones en caso que no sean ninguno de los valores anteriores

End Select
```

La instrucción *Case Else* nos permite controlar valores de *Expresión_prueba* que no coinciden con ninguno de los valores de Case.

Ejemplo:

En este ejemplo se calcula la bonificación por antigüedad en función del tiempo laborado por el empleado para la empresa.

```
Select Case Dias

        Case Is <= 365
                Antiguedad = 0

        Case Is <= 730
                Antiguedad = Salario * 0.03
```

```
        Case Is <= 1095
                Antiguedad = Salario * 0.05

        Case Is <= 1461
                Antiguedad = Salario * 0.07

        Case Is <= 1826
                Antiguedad = Salario * 0.09

        Case Is <= 2191
                Antiguedad = Salario * 0.1

        Case Is <= 2556
                Antiguedad = Salario * 0.11

        Case Is <= 2922
                Antiguedad = Salario * 0.12

        Case Is <= 3287
                Antiguedad = Salario * 0.13

        Case Is <= 3652
                Antiguedad = Salario * 0.14

        Case Is <= 4017
                Antiguedad = Salario * 0.15

        Case Is <= 4383
                Antiguedad = Salario * 0.155

        Case Is <= 4748
                Antiguedad = Salario * 0.16

        Case Is <= 5113
                Antiguedad = Salario * 0.165

        Case Is <= 5478
                Antiguedad = Salario * 0.17
                    .
                    .
                    .
        Case Is <= 7305
                Antiguedad = Salario * 0.195

        Case Else
                Antiguedad = Salario * 0.2

    End Select
```

- **La Función *InputBox*.**

Esta función muestra una ventana para que el usuario pueda teclear datos. Cuando se pulsa sobre aceptar, los datos entrados pasan a la variable a la que se ha igualado la función.

Sintaxis de InputBox.

```
InputBox(Mensaje, Título, Valor por defecto, Posición horizontal,
Posición Vertical, Archivo ayuda, Número de contexto para la ayuda).
```

Mensaje: Es el mensaje que se muestra en la ventana. Si desea poner más de una línea ponga *Chr(13)* para cada nueva línea.

Título: Es el título para la ventana *InputBox*. Es un parámetro opcional.

Valor por defecto: Es el valor que mostrará por defecto el cuadro donde el usuario entra el valor. Parámetro opcional.

Posición Horizontal: La posición X de la pantalla donde se mostrará el cuadro, concretamente es la posición para la parte izquierda. Si se omite el cuadro se presenta horizontalmente centrado a la pantalla.

Posición Vertical: La posición Y de la pantalla donde se mostrará el cuadro, concretamente es la posición para la parte superior. Si se omite el cuadro se presenta verticalmente centrado a la pantalla.

Archivo Ayuda: Es el archivo que contiene la ayuda para el cuadro. Parámetro opcional.

Número de contexto para la ayuda: Número asignado que corresponde al identificador del archivo de ayuda, sirve para localizar el texto que se debe mostrar. Si se especifica este parámetro, debe especificarse obligatoriamente el parámetro Archivo Ayuda.

- **La función MsgBox.**

Esta función muestra un mensaje en un cuadro de diálogo hasta que el usuario pulse un botón. La función devuelve un dato tipo *Integer* en función del botón pulsado por el usuario. A la hora de invocar está función, se permiten diferentes tipos de botones.

Sintaxis de *MsgBox*.

```
MsgBox(Mensaje, Botones, Título, Archivo de ayuda, contexto)
```

Mensaje: Obligatorio, es el mensaje que se muestra dentro del cuadro de diálogo.

Botones: Opcional. Es un número o una suma de números o constantes (vea tabla Valores para botones e Íconos), que sirve para mostrar determinados botones e iconos dentro del cuadro de diálogo. Si se omite este argumento asume valor 0 que corresponde a un único Botón, OK.

Título: Opcional. Es el texto que se mostrará en la barra del título del cuadro de diálogo.

Archivo de Ayuda: Opcional. Si ha asignado un texto de ayuda al cuadro de diálogo, aquí debe especificar el nombre del archivo de ayuda donde está el texto.

Context: Opcional. Es el número que sirve para identificar el texto al tema de ayuda correspondiente que estará contenido en el archivo especificado en el parámetro Archivo de ayuda.

Tabla para botones e íconos de la función MsgBox.

Constante	Valor	Descripción
VbOKOnly	0	Muestra solamente el botón **Aceptar**.
VbOKCancel	1	Muestra los botones **Aceptar** y **Cancelar**.
VbAbortRetryIgnore	2	Muestra los botones **Anular**, **Reintentar** e **Ignorar**.
VbYesNoCancel	3	Muestra los botones **Sí**, **No** y **Cancelar**.
VbYesNo	4	Muestra los botones **Sí** y **No**.
VbRetryCancel	5	Muestra los botones **Reintentar** y **Cancelar**.
VbCritical	16	Muestra el icono de **mensaje crítico**.
VbQuestion	32	Muestra el icono de **pregunta de advertencia**.
VbExclamation	48	Muestra el icono de **mensaje de advertencia**.
VbInformation	64	Muestra el icono de **mensaje de información**.
VbDefaultButton1	0	El primer botón es el predeterminado.
VbDefaultButton2	256	El segundo botón es el predeterminado.
VbDefaultButton3	512	El tercer botón es el predeterminado.
VbDefaultButton4	768	El cuarto botón es el predeterminado.
VbApplicationModal	0	Aplicación modal; el usuario debe responder al cuadro de mensajes antes de poder seguir trabajando en la aplicación actual.
VbSystemModal	4096	Sistema modal; se suspenden todas las aplicaciones hasta que el usuario responda al cuadro de mensajes.
VbMsgBoxHelpButton	16384	Agrega el botón Ayuda al cuadro de mensaje.
VbMsgBoxSetForeground	65536	Especifica la ventana del cuadro de mensaje como la ventana de primer plano.
VbMsgBoxRight	524288	El texto se aliena a la derecha.
VbMsgBoxRtlReading	1048576	Especifica que el texto debe aparecer para ser leído de derecha a izquierda en sistemas hebreo y árabe.

4) Funciones

Una función es lo mismo que un procedimiento con la salvedad que tiene un tipo y devuelve un valor al procedimiento o función que lo llama.

La estructura general sería:

```
Function Nombre_Función(párametro1 As Tipo, párametro2 As Tipo,..., párametroN As Tipo) As Tipo.

   Sentencias

End Function
```

Ejemplo:

```
Sub Sumar _ números()

   Dim x As Integer
   Dim n1 As Integer, n2 As Integer

   n1 = Val(InputBox("Entrar un número : ", "Entrada"))
   n2 = Val(InputBox("Entrar otro número : ", "Entrada"))
   x = Suma(n1, n2)
   ActiveCell.Value = x

End Sub

Function Suma(V1 As Integer, V2 As Integer) As Integer

   Dim Total As Integer
   Total = V1 + V2
   Suma = Total

End Function
```

El procedimiento *Sumar _ números* pide dos números a través de un *InputBox* pero es la función suma la que realiza la suma de los dos números y pasa el valor al procedimiento *Sumar _ números*, el que a su vez manda el resultado a la celda activa del libro de Excel.

Una función definida en Visual Basic puede luego ser usada en la hoja de cálculo de Excel de la misma forma en que usaría la función suma, promedio, producto o cualquier otra función propia de Excel. Para eso la función se debe crear en un libro el que luego se debe guardar como Complemento de Excel en la ventana Guardar como.

Para agregar el complemento de manera que este disponible en cualquier libro de Excel se debe hacer lo siguiente:

Teniendo un libro abierto dar clic en el botón office/ Opciones de Excel/ Complementos/ Ir / Examinar/ seguir la ruta en donde se guardó el complemento/ seleccionar el complemento/ Aceptar.

Haciendo clic en insertar función, puede buscar la función creada en la categoría Definidas por el usuario.

Ejemplo:
La siguiente función calcula la cuota laboral que aporta un trabajador en el régimen integral al Seguro Social.

```
Function INSS (Salario As Double) As Double
   If salario < 37518 Then
      INSS = Salario * 0.0625
   Else
      INSS = 37518 * 0.0625
   End if
End Function
```

- **Llamar a un procedimiento**

Para llamar un procedimiento se utiliza la instrucción ***Call***. Su estructura es la siguiente:

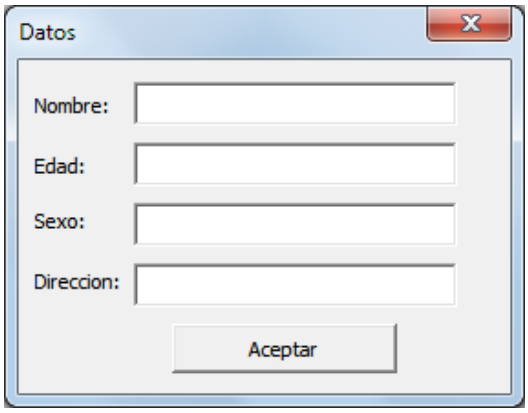
```
   Call Nombre_procedimiento
```

Un procedimiento que llama a otro en general tiene la siguiente estructura:

```
Sub Procedimiento_llamador
   Sentencias
   Call Procedimiento_llamado
   Sentencias
End Sub
```

Por ejemplo si en una base de datos en Excel deseamos agregar un nuevo registro necesitamos un procedimiento que nos ubique en la primera celda vacía de la hoja de Excel y otro con el que podamos guardar el registro. Esto lo podemos hacer en un mismo procedimiento pero si necesitamos la misma operación (encontrar la primer celda vacía) en otros procedimientos lo mejor es definir ese procedimiento y luego simplemente invocarlo, de esta manera nos evitamos estar repitiendo el mismo código en todos los procedimientos. Hay que aclarar que esto depende del gusto del programador y de la complejidad del programa y por tal razón no siempre se puede o debe llamar un procedimiento, en ese caso hay que escribir todo el código en un mismo procedimiento.

En el siguiente ejemplo se pide el nombre, edad, sexo y dirección de una persona y luego se guarda en la hoja 1 del libro de Excel. Para introducir los datos diseñamos el formulario que se muestra a continuación.



Dando doble clic sobre el botón Aceptar, en el Editor de Visual Basic, escribimos el código de los procedimientos que realizan las dos tareas antes señaladas.

```
Private Sub Guardar_Clic()

Dim Nombre, Dirección, Sexo As String
Dim Edad As Integer

   Nombre = TextBox1.Value
   Edad = TextBox2.Value
   Sexo = TextBox3.Value
   Dirección = TextBox4.Value

Call Buscar_celdas_vacias

With ActiveCell
     .Value = Nombre
     .Offset(0, 1).Value = Edad
     .Offset(0, 2).Value = Sexo
     .Offset(0, 3).Value = Dirección
End With

End Sub

Sub Buscar_celdas_vacias()

Hoja1.Range("A2").Activate

   Do While Not IsEmpty(ActiveCell)
      ActiveCell.Offset(1, 0).Activate
   Loop

End Sub
```

Hacemos que el procedimiento *Buscar_celdas_vacias* inicie en la celda "A2" porque suponemos, como en realidad debe ser, que la fila 1 es usada para el encabezado de columna.

- **Parámetros**

El procedimiento *Buscar_celdas_vacias* tal y como lo definimos en el procedimiento del ejemplo anterior presenta el inconveniente de que únicamente funciona si vamos a introducir los datos en la hoja 1 empezando en la celda A2 lo que limita su utilidad. Este problema lo podemos solucionar usando parámetros.

Los parámetros son el mecanismo por el cual un procedimiento puede pasarle valores a otro y de este modo condicionar las acciones que realiza, de modo que el procedimiento llamado gana flexibilidad y puede ser usado en cualquier hoja de cálculo e inicie el recorrido en cualquier celda.

Usando parámetros la sintaxis de llamada de un procedimiento queda así:

```
Call Nombre_procedimiento (parámetro1,  parámetro2, …, parámetroN)
```

La sintaxis del procedimiento llamado es la siguiente:

```
Sub Nombre_procedimiento(parámetro1 As Tipo, parámetro2 As Tipo, ..., parámetroN As Tipo)
```

Modificando el código de nuestro programa del ejemplo anterior de la siguiente manera, el procedimiento *Buscar_celdas_vacias* puede ser llamado por otros procedimientos en los que se necesite encontrar la primera celda vacía cualquier hoja empezando en cualquier celda.

```
Private Sub Guardar_Clic()

  Dim Nombre, Dirección, Sexo As String
  Dim Edad As Integer
  Nombre = TextBox1.Value
  Edad = TextBox2.Value
  Sexo = TextBox3.Value
  Dirección = TextBox4.Value

  Call Buscar_celdas_vacias("Hoja2", "A2")

  With ActiveCell
    .Value = Nombre
    .Offset(0, 1).Value = Edad
    .Offset(0, 2).Value = Sexo
    .Offset(0, 3).Value = Dirección
  End With

End Sub
```

```
Sub Buscar_celdas_vacias(hoja As String, celda_inicial As String)

Worksheets(hoja).Activate
ActiveSheet.Range(celda_inicial).Activate

   Do While Not IsEmpty(ActiveCell)
     ActiveCell.Offset(1, 0).Activate
   Loop

End Sub
```

5) Detección de errores y depuración de programas

A medida que los programas van creciendo la probabilidad de cometer errores también va creciendo. Los errores se clasifican normalmente en tres categorías.

Errores en tiempo de compilación. Son los típicos errores que impiden hacer funcionar el programa debido, por ejemplo, a errores de sintaxis en las instrucciones, llamadas a funciones que no existen o llamadas con el tipo o el número de parámetros incorrectos, etc. Este tipo de errores no dan demasiados problemas, primero porque el compilador avisa de donde se han producido y luego porque simplemente revisando la sintaxis se solucionan rápidamente.

Errores en tiempo de ejecución. Estos errores se producen por una mala programación del código al no haber previsto determinados casos concretos o especiales, como por ejemplo intentar abrir un archivo que no existe, imprimir sin comprobar que la impresora está conectada, definir mal la dimensión de un *array* o intentar acceder a miembros que no existen, etc. Cuando se produce este tipo de errores se detiene la ejecución del programa y normalmente se informa del tipo de error que se ha producido.

Errores de función. Son los más complicados de detectar ya que ni se detectan en la fase de ejecución, ni provocan la detención del programa, son debidos a la incorrecta programación de algún proceso y como resultado se obtienen datos erróneos. Errores de este tipo son cálculos mal hechos, bucles infinitos, devolución de valores incorrectos, etc. Como ni los detecta el compilador, ni provocan la interrupción del programa deben revisarse a mano, y claro, si el programa es extenso y trata muchos datos, su detección puede resultar dificultosa. Visual Basic, y todos los entornos de programación incorporan herramientas para facilitar la detección de este tipo de errores, son las herramientas de depuración.

- **Herramientas de depuración**

Antes de iniciar la depuración de un programa lo ideal es activar la barra de depuración (Ver/ Barra de Herramientas/Depuración).

Modo Ejecución paso a paso por instrucciones.

El modo paso a paso permite la ejecución del programa instrucción por instrucción, de esta forma es posible ver que el funcionamiento del programa es el correcto, es decir que la ejecución de instrucciones sigue los pasos que se habían previsto. Para ejecutar un procedimiento paso a paso, sólo debe ir pulsando sobre el botón , activar la opción de menú **Depuración /Paso a Paso por Instrucciones** o ir pulsando la tecla **F8**, que seguramente es lo más cómodo.

Cuando esté ejecutando paso a paso puede utilizar los botones siguientes para llevar a cabo determinadas acciones.

Sirve para detener la ejecución del programa.

Sirve para ejecutar el resto del programa.

Sirve para ejecutar todo un procedimiento. Cuando en la ejecución de un procedimiento, se llega a una línea que llama a otro procedimiento o función, pulsando este botón se puede provocar la ejecución de todo el código de esta función para luego continuar con el modo paso a paso.

Sirve para ejecutar todas las instrucciones del procedimiento activo y volver (o terminar).

El modo Interrupción

En programas largos resulta fastidioso tener que ejecutarlos paso a paso, sobre todo si sabemos que el error se produce en una parte avanzada del programa. El modo interrupción, permite la ejecución del programa hasta una instrucción determinada para, a partir de esta, ejecutar paso a paso y así poder detectar el error.

Definir puntos de interrupción.

1. Sitúe el cursor sobre la instrucción en la cual debe detenerse el programa para continuar paso a paso.

2. Pulse sobre el botón . También puede activar la opción **Depuración/ Alternar punto de interrupción**, pulsar la tecla F9 o bien hacer un clic en la parte izquierda

de la ventana del módulo (la franja vertical en color gris). Siga los mismos pasos para desactivar un punto de interrupción.

- **Solucionar los errores.**

Todo lo dicho anteriormente no serviría de gran cosa si no fuera posible revisar los valores que las variables van cogiendo a medida que vamos avanzando o si no tuviéramos ocasión de evaluar las expresiones del programa. En las siguientes secciones veremos como llevar a cabo estas acciones.

Inspecciones rápidas de variables.

Estas opciones sirven para revisar el valor de las variables a medida que se va ejecutando el programa.

Para ver los valores que van tomando las variables es conveniente tener visible la **Ventana de inspección**, para activarla haga clic en **Ver/ Ventana de Inspección** o pulse sobre el botón  lo que despliega el panel en la parte de abajo en la ventana.

Añadir una variable a la ventana de inspección.

Aunque no es necesario estar ejecutando el programa en modo paso a paso, es conveniente.

1. Seleccione la variable que desee añadir a la ventana haciendo un clic sobre ella.

2. Pulse sobre el botón  , también puede activar **Depuración/ Inspección rápida** o pulsar la combinación **MAYUSC+F9**. Aparece un cuadro de diálogo donde se muestra el valor actual de la variable. Si no está ejecutando el programa paso a paso, aparecerá el valor **Fuera de Contexto**.

3. Pulse sobre el botón **Agregar** para añadir la variable a la ventana de inspección. Debe tener en cuenta que para revisar las variables las expresiones que les asignan valores deben de ejecutarse al menos una vez.

Cuando ejecuta el programa paso a paso, si sitúa el puntero de ratón sobre una variable se muestra el valor de la misma.

Borrar una variable de la ventana de Inspección.

Sólo debe seleccionarla en la ventana de inspección y pulsar sobre la tecla **SUPR**.

Modificar el valor de una variable en tiempo de ejecución.

A veces resulta interesante cambiar el valor de alguna variable cuando se está ejecutando el programa, para ver que ocurre si coge determinados valores, para terminar un bucle, etc.

Esto lo puede hacer de la siguiente manera.

1. Sitúe un punto de interrupción en la línea de la variable de inspección.
2. Agregue a la ventana de inspección (si no está) la variable que se esta inspeccionando.
3. Ejecute el programa, al detenerse, observe en la **Ventana de Inspección** que la variable *Inspeccionada* tiene un valor que ahora cambiaremos.
4. Haga doble clic sobre el valor de *inspeccionada* dentro de la ventana de inspección.
5. Borre el valor que tiene, cámbielo por otro y pulse **ENTER**.
6. Ahora puede continuar con la ejecución normal del programa.

Expresiones de Revisión.

Además de permitir añadir una variable o expresión dentro de la **Ventana Inmediato**, una **Expresión de Revisión** permite interrumpir la ejecución del programa cuando una variable coge determinado valor. Piense que muchas veces un programa deja de funcionar, o funciona mal cuando una variable coge determinados valores. Con una expresión de revisión, podremos detener la ejecución del programa cuando una variable contiene determinado valor, luego podremos continuar con la ejecución paso a paso para ver que ocurre a partir de este punto.

La Ventana Inmediato.

Es otra forma de inspeccionar variables cuando el programa está en modo interrupción (ejecutándose paso a paso), pero además, ofrece la posibilidad de cambiar valores de las variables e incluso ejecutar o evaluar expresiones. Para ver el valor de una variable en la ventana inmediato debe anteponerle el signo de interrogación (?) y luego pulsar Enter.

Para activar la ventana Inmediato, active opción Ver/Inmediato, o pulse la combinación CONTROL+G.

La instrucción Debug.Print.

Esta instrucción, que se utiliza directamente sobre el código del programa, permite ver todos los valores que ha ido recogiendo una variable o expresión durante la ejecución del programa. Los valores se mostrarán en la ventana Inmediato una vez finalizado el programa. Esta expresión resulta útil en una fase avanzada de depuración ya que permite ir

viendo la evolución de una variable o expresión sin necesidad de poner puntos de interrupción.

Evidentemente cuando el programa esté listo deben de sacarse.

- **Errores de ejecución**

El objeto Err.

Siempre que se produce un error en tiempo de ejecución Visual Basic genera (o dispara como dicen algunos autores) un objeto tipo **Err**, estudiaremos dos propiedades de este objeto **Number** y **Description**.

Number: Es un número indicativo sobre el tipo de error que se ha producido, dicho de otra forma, cuando Visual Basic dispara un objeto **Err** en tiempo de ejecución, podemos revisar su propiedad **Number** para saber que es lo que ha causado el error.

Description: Es el mensaje asociado al número de error y es una cadena de texto que describe brevemente el error. Implementando una rutina de error hacemos que se muestre el número de error y su descripción, insistimos que es lo que hace Visual Basic por si solo, pero luego veremos como gracias al número de error podremos determinar la causa del error y controlar la ejecución del programa como más convenga.

Para ver el número y descripción se debe escribir las siguientes líneas de código en el programa.

```
On Error GoTo Tratar_Errores

Sentencias

Tratar_Errores:
  MsgBox ("Número de Error : " & Err.Number & Chr(13) & "Descripción : "
  & Err.Description)
```

Al producirse un error en tiempo de ejecución se mostraría un *MsgBox* que muestra el número y descripción del error que se ha producido.

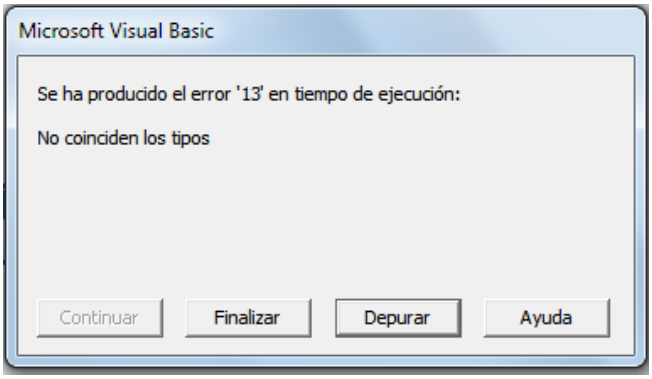
Es imposible excluir del todo los errores en los programas. Si además, y como es de desear, el programa será utilizado por usuarios que no han tenido nada que ver en el proceso de desarrollo e implementación posiblemente se producirán errores debido a su incorrecta utilización. Estos errores son los que se deben prevenir. Errores de este tipo son, por ejemplo, intentar acceder a un archivo inexistente, entrar valores incorrectos a través de un cuadro de diálogo o formulario (datos tipo *String* cuando se requieren números, etc.). También entrarían en este tipo de errores aquellos casos excepcionales pero que deben ser previstos por el programador, como por ejemplo que se llene la unidad de disco, que la impresora se quede sin papel, etc.

Visual Basic y la mayoría de lenguajes de programación permiten implementar rutinas de tratamiento de errores cuya finalidad es interceptar determinados tipo de errores que se producen en tiempo de ejecución.

La finalidad de estas rutinas es que el programa no se detenga, o al menos si se detiene, informar sobre la posible causa del error e intentar controlarlo de alguna forma.

6) Rutina de tratamiento de errores

Generalmente una rutina de tratamiento de errores reacciona ante casos esperados por el programador. Por ejemplo, podría ser que el usuario entrara caracteres en lugar de números por lo que el programa generaría el error siguiente.



Mediante una rutina de tratamiento de errores podemos informar del error que se ha producido y direccionaremos la ejecución del programa hacia donde interese. En Visual Basic el tratamiento de errores es una parte normal de la ejecución del programa. La instrucción para el tratamiento de errores es **ON ERROR GOTO *línea,*** línea es una etiqueta o marca de línea que indica hacia donde debe dirigirse el programa en caso de error. El formato general de un procedimiento o función donde se implementa una rutina de tratamiento de errores es la siguiente.

```
Sub Nombre()

On Error GoTo Etiqueta

'Instrucciones del procedimiento

Exit Sub 'Salir del procedimiento

Etiqueta:

'Instrucciones de tratamiento de error

End Sub
```

Con **On Error GOTO *Etiqueta***, indicamos al programa que en caso que se produzca un error en tiempo de ejecución vaya a ejecutar las líneas que siguen a la etiqueta o marca de línea *Etiqueta*.

Observe que antes de la etiqueta está la instrucción ***Exit Sub***, sirve para que si el procedimiento se ha desarrollado correctamente, salga en ese punto, tenga en cuenta que si no se pusiera esta línea la ejecución continuaría secuencialmente y se ejecutarían las líneas de la rutina *Etiqueta*.

Si la rutina de tratamiento de errores se implementa en una función debe poner ***Exit Function*** en lugar de ***Exit Sub***.

- **Continuar el programa después de la ejecución de la rutina de tratamiento de errores.**

En la mayoría de los casos, las rutinas de tratamiento de errores son para que el programa no se detenga.

El proceso habitual cuando se produce el error es tratar de corregirlo y continuar la ejecución del programa. La continuación del programa se consigue mediante la instrucción ***Resume***.

Esta instrucción da tres posibilidades, Ejecutar otra vez la instrucción que ha causado el error, continuar por la instrucción siguiente a la que ha causado el error o dirigir la ejecución hacia una línea marcada con una etiqueta.

- **Resume**.

Ejecutar de nuevo la instrucción que ha causado el error. Esto podría ser volver a preguntar de nuevo el valor en caso que se estén ingresando datos desde el teclado y no coincidan los tipos.

La rutina de tratamiento de errores cuando se necesita que el valor introducido sea numérico quedaría.

```
Tratar_Errores:
   MsgBox ("Debe entrar un valor numérico")
Resume
```

- **Resume Next**

Continuar el programa por la instrucción siguiente a la que ha causado el error.

```
Tratar_Errores:
   MsgBox("Debe entrar un valor numérico")
Resume Next
```

En este caso el programa informaría de la causa del error pero continuaría la ejecución por la línea siguiente a la que ha causado el error. No es una buena solución para algunos casos pero los hay en los que es la mejor.

- **Resume Etiqueta**

Continuar por la instrucción que sigue a la *ETIQUETA*.

Haremos que el programa vuelva al principio cuando se produzca un error. Para ello debemos poner una etiqueta o marca de línea mediante la cual indicaremos en que punto debe continuar la ejecución del programa. La etiqueta se puede colocar al principio del procedimiento, aunque evidentemente no siempre tiene que empezar desde el principio, puede dirigirse la ejecución hacia donde más convenga.

```
Sub Nombre_procedimiento()
On Error GoTo Tratar_Errores 'Etiqueta que dirige el programa en caso de
error
Dim var1 As Tipo
Dim var2 As Tipo
   .
   .
   .
Dim varN As Tipo

Inicio: 'Aquí vuelve el programa si se produce un error
Sentencia1
Sentencia1
   .
   .
   .
SentenciaN

Exit Sub

Tratar_Errores:
  MsgBox ("Debe entrar un valor numérico")'O lo que se considere

Resume Inicio

End Sub
```

Cuando se produzca un error el programa se dirigirá hacia la etiqueta *Tratar_Errores*, se ejecutará la o las sentencias de la etiqueta, por ejemplo pedir que se introduzca un valor numérico, luego *Resume Inicio* manda a ejecutar el programa en el punto que hemos marcado.

Como es de suponer en un programa puede producirse más de un tipo de error, entonces el programador deberá prever la mayor parte posible, gracias a que cada error tiene un número asignado, utilizando la instrucción *Select Case* y dejando siempre un *Case Else* para cualquier caso no previsto.

```
Sub Nombre_procedimiento()
On Error GoTo Tratar_Errores 'Etiqueta que dirige el programa en caso de
error
Dim var1 As Tipo
Dim var2 As Tipo
   .
   .
   .
Dim varN As Tipo

Inicio: 'Aquí vuelve el programa si se produce un error
Sentencia1
Sentencia1
   .
   .
   .
SentenciaN

Exit Sub

Tratar_Errores:
Select Case Err.Number
  Case 13:
    MsgBox ("Debe introducir valores numéricos")

  Case 6:
    MsgBox ("El valor es demasiado grande o demasiado pequeño. " & _
    El número debe estar comprendido entre -32768 y 32767")

  Case Else
    MsgBox ("Error no previsto. Llame al responsable facilitando la " &_
    "información que sigue " & Chr(13) & "Número de Error : " & _
    Err.Number & Chr(13) & "Descripción : " & Err.Description)

Exit Sub ' Acción a realizar si el tipo de error es cualquier otro

End Select

Resume Inicio

End Sub
```

En el ejemplo genérico mostrado, con la instrucción *Select Case* tratamos dos errores en concreto: 13 que se produce cuando los tipos no coinciden (texto en lugar de números) y el 6 que se produce cuando se da un desbordamiento (Un desbordamiento ocurre cuanto intenta realizar una asignación que supera las limitaciones del destino de la asignación), al final es deseable dejar una opción para cuando se produzca otro error (*Case Else*), en este caso el programa muestra en un cuadro de diálogo el número y la descripción del error. Esta información es necesaria para que el programador pueda dar solución al error.

Cuando un procedimiento llama a otro o a una función es posible que se produzca un error no en el procedimiento llamador sino en el o los procedimientos o funciones llamadas, si

tenemos una rutina de tratamiento de errores en el procedimiento llamador el programa no se detendrá porque se pasaría a ejecutar dicha rutina, el problema aquí radica en que *Err.Number* recoge únicamente el número del error que se produzca en su respectivo procedimiento, así que si el problema se da en el procedimiento llamado la rutina de tratamiento de errores no ayudaría de mucho. Muchas veces la solución a este tipo de errores es definir una variable global, la que recogerá el número de error que se produzca en cualquier procedimiento (recuerde que una variable global no esta asignada a ningún procedimiento y por tanto puede ser llamada por cualquiera) y luego una simple rutina de tratamiento de errores en el procedimiento o función llamada soluciona el inconveniente.

- **Definir errores propios.**

Visual Basic incorpora la instrucción **Error** con la que es posible definir o generar errores propios. Puede que se esté preguntando para que quiera definirse errores propios, dicho de otra forma, para que provocar errores propios. Bueno, pues por ejemplo, para un mayor control en programas largos y complejos, para validar datos, y algunos procesos más.

Para provocar un error simplemente tiene que utilizar la instrucción **Error**(*Número_De_Error*), *Número_De_Error* debe ser un valor comprendido entre 0 y 65535, es decir debe estar dentro del rango de errores definidos por Visual Basic, ahora bien, utilice siempre valores altos ya que si no, corre el riesgo de modificar un valor ya establecido por Visual Basic. Le recomendamos que empiece a partir del valor 65535 y vaya bajando, estos valores tan elevados nunca serán utilizados por Visual Basic.

V. HERRAMIENTAS DE VISUAL BASIC PARA APLICACIONES

1) Ventanas del Editor de Visual Basic

El Editor de Visual Basic está compuesto por las siguientes partes:

- Barra de menú
- Barras de herramientas
- Ventana Explorador de proyectos.

Presenta un diagrama de árbol que contiene cada libro de trabajo que está actualmente abierto en Excel. Cada libro de trabajo es un proyecto. Si esta ventana no esta abierta hay que presionar Control+R. Para esconder la ventana, hay que hacer clic en el botón Cerrar de su barra de título.

Ventana código.

Una ventana de código, o ventana de módulo, contiene un código VBA. Cada elemento de un proyecto tiene asociada una ventana de código. Para visualizar una ventana de código

para un objeto, hay que hacer doble clic sobre el objeto en la ventana del explorador de proyectos.

Ventana Inmediato.

Esta ventana es útil para ejecutar instrucciones de VBA directamente, probar las instrucciones y limpiar el código. Una forma de abrir esta ventana presionar la combinación de teclas Control+G, para ocultarla basta hacer clic sobre el botón Cerrar de su barra de título.

- **Guardar un código VBA**

En general una ventana de código puede soportar cuatro tipos de código:

Procedimientos Sub: Conjuntos de instrucciones que ejecutan alguna acción.

Procedimientos Function: Es un conjunto de instrucciones que devuelven un solo valor.

Procedimientos Property: Son procedimientos especiales que se usan en módulos de clase.

Declaraciones: Es información acerca de una variable que se le proporciona a VBA.

Un solo módulo de VBA puede guardar cualquier cantidad de procedimientos Sub, procedimientos *Function* y declaraciones.

- **Ejecutar un código VBA**

Para ejecutar un procedimiento desde el editor de Visual Basic hay que asegurarse que el cursor está situado en alguna parte del procedimiento a ejecutar, luego puede optar por cualquiera de estas opciones.

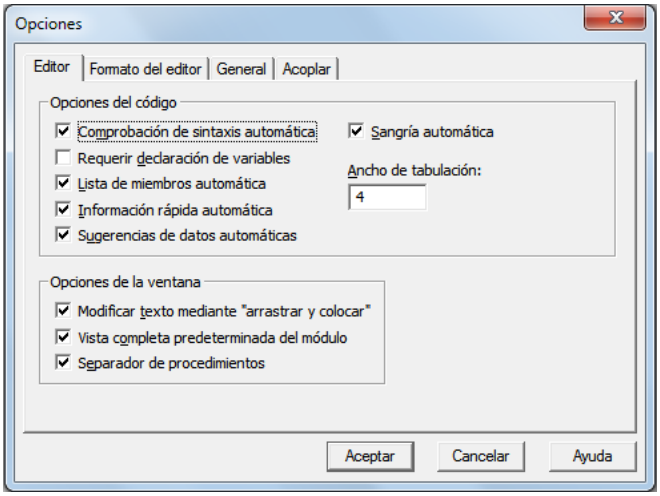
- Presionar F5
- Seleccionar la opción Ejecutar/ Ejecutar Sub/ UserForm de la Barra de Menú
- Dar clic en el botón Ejecutar Sub/ UserForm de la Barra de Herramientas Estándar

2) <u>Personalizar el entorno del editor de VBA</u>

Dando clic en la ficha Herramientas y luego en Opciones de la Barra de Menú se activa la ventana siguiente que contiene cuatro fichas (Editor, Formato del editor, General y Acoplar) las que permiten personalizar el entorno de Visual Basic.

➢ LA FICHA EDITOR

Especifica las configuraciones de las ventanas de Código y de Proyecto.



Opciones del código

- Comprobación automática de sintaxis: determina si Visual Basic debería verificar automáticamente la sintaxis correcta después de introducir una línea de código.

- Requerir declaración de variables: determina si se necesitan declaraciones de variables explícitas en los módulos. Al seleccionar esto, se agrega la instrucción Opción explícita a las declaraciones generales de cualquier módulo nuevo.

- Lista de miembros automática: muestra una lista que contiene información que completaría de forma lógica la instrucción en el punto actual de inserción.

- Información rápida automática: muestra información sobre funciones y sus parámetros según escribe.

- Sugerencia de datos automática: muestra el valor de la variable sobre la que se sitúa el puntero. Sólo está disponible en modo de interrupción.

- Sangría automática: permite tabular la primera línea de código; todas las líneas posteriores comenzarán en la ubicación del tabulador.

- Ancho del tabulador: establece la anchura del tabulador, que puede variar de 1 a 32 espacios; el valor predeterminado es 4 espacios.

Opciones de la ventana

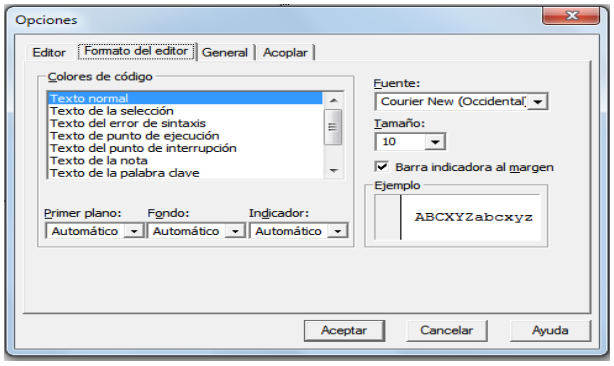
- Modificar texto con arrastrar y colocar: permite arrastrar y colocar elementos dentro del código actual y de la ventana de Código a la ventana Inmediato o Inspección.

- Vista completa predeterminada del módulo: establece el estado predeterminado de módulos nuevos para permitirle mirar en los procedimientos en la ventana de Código como lista desplazable única o sólo como un procedimiento a la vez. No afecta a la forma en que se ven los módulos abiertos actuales.

- Separador de procedimientos: permite mostrar u ocultar barras separadoras que aparecen al final de cada procedimiento en la ventana de Código.

➢ **FORMATO DEL EDITOR**

Especifica la presentación del código de Visual Basic.



Colores de código

Determina los colores de primer plano y de fondo a utilizar en el tipo de texto seleccionado en el cuadro de Lista.

- Lista texto de color: enumera los elementos de texto que tienen colores personalizables.

- Primer plano: especifica el color de primer plano en el texto seleccionado en la lista de texto de color.

- Fondo: especifica el color de fondo en el texto seleccionado en la lista de texto de color.

- Indicador: especifica el color del indicador de margen.

Fuente

Especifica la fuente a utilizar en todo el código.

Tamaño

Especifica el tamaño de la fuente a utilizar en todo el código.

Barra del indicador de margen

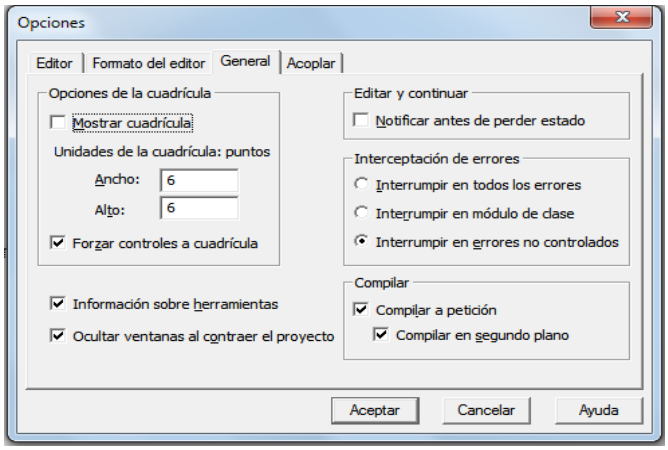
Hace visible o invisible la barra del indicador de margen.

Muestra

Presenta texto de muestra para las configuraciones de fuentes, tamaños y colores.

- **FICHA GENERAL**

Especifica las configuraciones, el tratamiento de errores y la compilación de configuraciones para el proyecto actual de Visual Basic.



Opciones de la cuadrícula

Determina la presentación del formulario cuando se ha editado.

- Mostrar cuadrícula: determina si se debe mostrar la cuadrícula.
- Unidades de cuadrícula: muestra las unidades de cuadrícula utilizadas en el formulario.
- Ancho: determina la anchura de las celdas de la cuadrícula en un formulario (2 a 60 puntos).

- Alto: determina la altura de las celdas de la cuadrícula en un formulario (2 a 60 puntos).
- Alinear controles en cuadrícula: coloca automáticamente los bordes exteriores de los controles en las líneas de la cuadrícula.

Información sobre herramientas

Muestra información sobre herramientas en los botones de la barra de herramientas.

Ocultar ventanas al contraer el proyecto

Determina si las ventanas de proyecto, *UserForm*, objeto o módulo se cierran automáticamente cuando se oculta un proyecto en el Explorador de proyectos.

Editar y continuar

Notificar antes de la pérdida del estado: determina si se recibirá un mensaje notificándole que la acción solicitada provocará que todas las variables de nivel de módulo se restauren para el proyecto en curso.

Interceptación de errores

Determina la forma de tratar los errores en el entorno de desarrollo de Visual Basic. La configuración de esta opción afecta a todas las instancias de Visual Basic que comenzaron después de cambiar la configuración.

- Interrumpir en todos los errores: cualquier error que motive que el proyecto entre en modo de interrupción, sin importar si está activada una opción de tratamiento de errores ni si el código está en módulo de clase.

- Interrumpir en módulo de clase: cualquier error no controlado que se ha producido en clase de módulo que motive que el proyecto entre en modo de interrupción en la línea de código del módulo de clase que produjo el error.

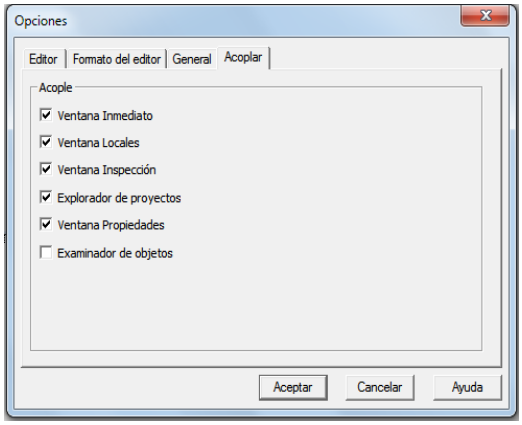
- Interrumpir en errores no controlados: si está activada una opción de tratamiento de errores, se intercepta el error sin entrar en modo de interrupción. Si no está activada una opción de tratamiento de errores, el error motiva que el proyecto entre en modo de interrupción. Un error no controlado en módulo de clase, sin embargo, motiva que el proyecto entre en modo de interrupción en la línea de código que invocó el procedimiento incorrecto de clase.

Compilar

- Compilar a petición: determina si un proyecto se ha compilado por completo antes de que comience o si el código se ha compilado según sea necesario, permitiendo que la aplicación comience lo antes posible.

- Compilar en segundo plano: determina si se debe utilizar el tiempo en espera durante el tiempo de ejecución para finalizar la compilación en segundo plano del proyecto. Compilar en segundo plano puede mejorar la velocidad del tiempo de ejecución.

➢ **FICHA ACOPLAR**



Permite elegir qué ventanas desea que sean acoplables. Una ventana es acoplable cuando se adjunta o se "ancla" a un borde de otra ventana acoplable o de la aplicación. Cuando desplaza una ventana acoplable, "salta" a la ubicación elegida. Una ventana no es acoplable cuando puede desplazarla a cualquier lugar de la pantalla y dejarla ahí.

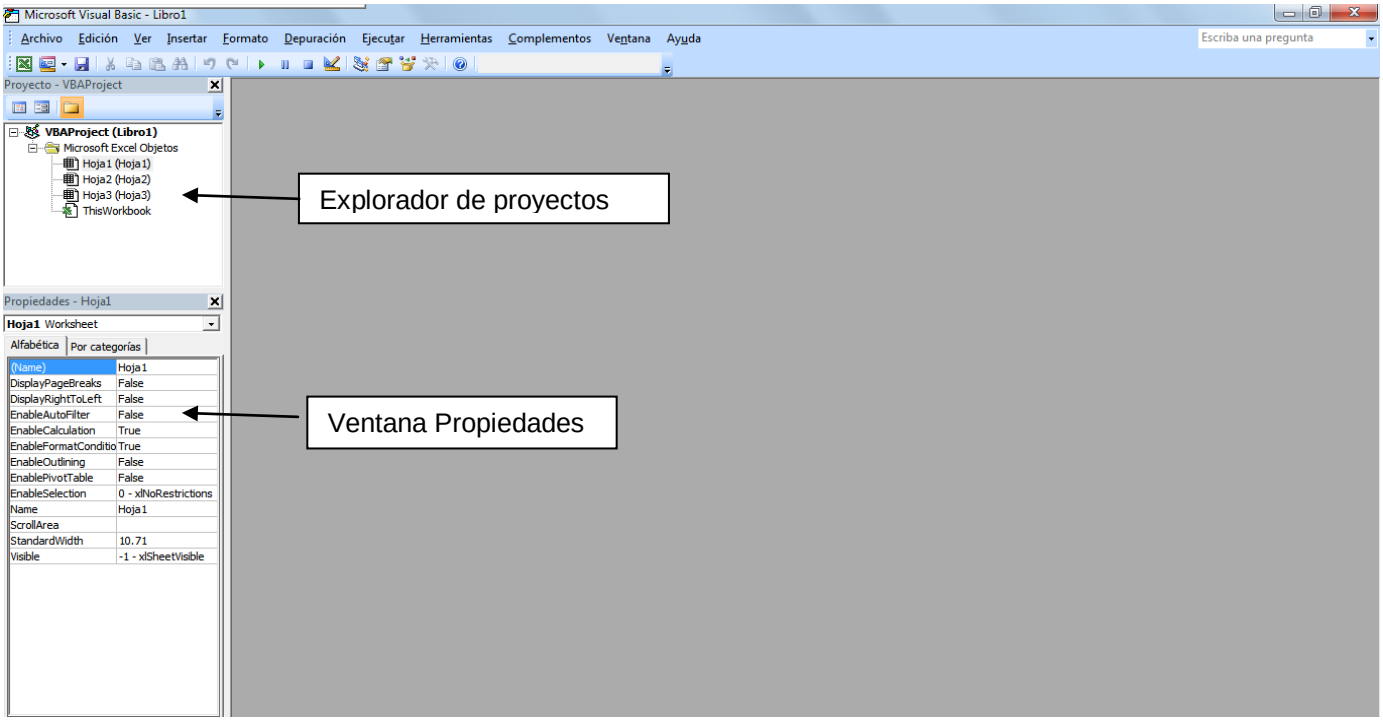
Seleccione las ventanas que desea que sean acoplables y elimine las que no. Cualquiera, ninguna o todas las ventanas de la lista pueden acoplarse.

3) Creación de formularios

Hasta el momento hemos hablado de la forma en la que se puede generar los códigos de un programa pero no de cómo crear la interfaz de usuario es decir, crear formularios y programarlos.
Bueno un formulario es una ventana en la que se diseña la interfaz por medio de controles, los que responden a sucesos que nosotros programamos. Todo esto se encuentra dentro de Visual Basic.

A continuación mostraremos como crear y personalizar un formulario:

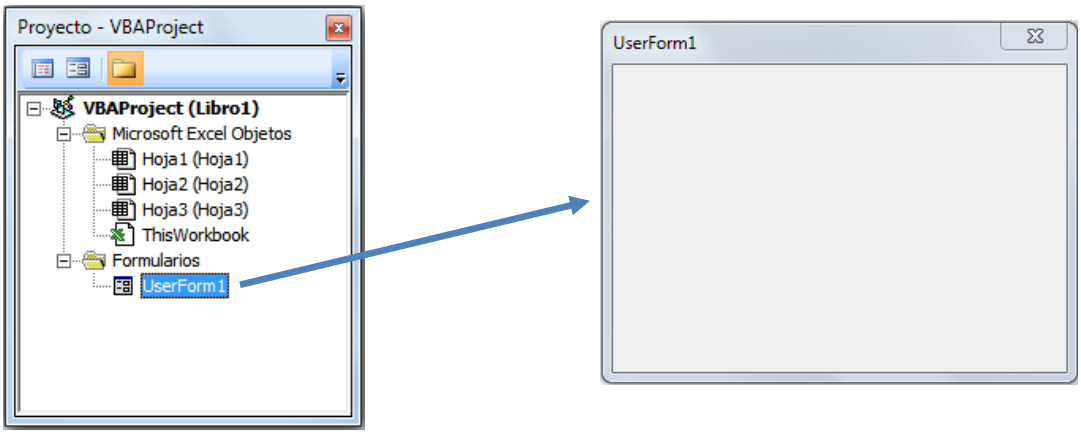
Presione La Teclas **Alt + F11,** para entrar al editor de **Visual Basic** y se despliega la siguiente pantalla.



Active las siguientes opciones:

- De clic en el **Menú Ver** y elija la opción **Explorador de Proyectos**
- De clic en el **Menú ver** y elija la opción **Ventana Propiedades**

Del **Menú Insertar** elija la Opción **UserForm.** Esto inserta el Formulario que programaremos con controles. En el **Explorador de Proyecto** se observara que se inserto el **UserForm**.



También cuando de clic en el Formulario **USERFORM1** se debe de activar **el Cuadro de Herramientas**, si no se activa de clic en el **Menú Ver** y elija la opción **Cuadro de Herramientas**.

En el cuadro de herramienta encontramos los distintos controles que se utilizan para dar vida a un programa automatizado. Los controles son aquellos que aparecen "cargados por defecto" en el cuadro de herramientas cuando le damos insertar **USERFORM1**

Label (Etiqueta): 

Muestra la información, los títulos o los subtítulos que deseemos.

Un control *Label* en un formulario muestra un texto descriptivo como títulos, leyendas, imágenes o breves instrucciones. Por ejemplo, las etiquetas para una libreta de direcciones podrían incluir un control *Label* para el nombre, la calle o la ciudad. Un control *Label* no muestra valores de orígenes de datos ni expresiones; es siempre independiente y no cambia cuando se mueve de un registro a otro.

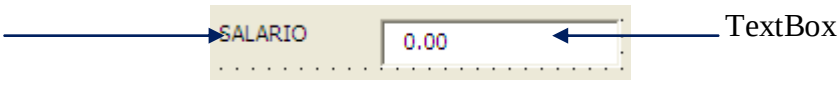
La propiedad predeterminada de un control *Label* es *Caption* y el evento predeterminado de un control *Label* es *Clic*.

TextBox (Caja de texto): 

Se utiliza para el ingreso y validación de datos o información.
El control *TextBox* es el control utilizado habitualmente para mostrar información escrita por un usuario. También puede mostrar un conjunto de datos como una tabla, una consulta, una hoja de cálculo o el resultado de un cálculo. Si un control *TextBox* es dependiente de un origen de datos, al cambiar el contenido del control *TextBox* también cambia el valor del origen de datos dependiente.

El formato aplicado a cualquier fragmento de texto en un control *TextBox* afectará a todo el texto del control. Por ejemplo, si cambia la fuente o el tamaño del punto de cualquier carácter del control, el cambio afectará a todos los caracteres del control.

La propiedad predeterminada de un control *TextBox* es *Value* y el evento predeterminado de un control *TextBox* es *Change*.

Label  TextBox

CommandButton (Botón de Comando): 

Este control permite encapsular procedimientos para que sean utilizados por el usuario con solo hacer clic sobre él.

La macro o el procedimiento de evento asignado al evento *Clic* del control *CommandButton* determina qué hace el control *CommandButton*. Por ejemplo, puede crear un control *CommandButton* que abra otro formulario. También puede mostrar un texto, una imagen o ambos en un control *CommandButton*.

La propiedad predeterminada de un control *CommandButton* es *Value* y el evento predeterminado de un control *CommandButton* es *Clic*.

CheckBox (Casilla de verificación): 

Permite que se ejecute un procedimiento dependiendo de la verificación del *Checkbox*.

Utilice un control *CheckBox* para ofrecer al usuario la posibilidad de elegir entre dos posibles valores como *Sí*/*No*, *Verdadero*/*Falso*, o *Activado*/*Desactivado*. Cuando el usuario selecciona un control *CheckBox*.

Éste muestra una marca especial y su valor actual es *Sí*, *Verdadero*, o *Activado*; si el usuario no selecciona el control *CheckBox*, éste está vacío y su valor es *No*, *Falso*, o *Desactivado*.

Si un control *CheckBox* es dependiente de un origen de datos, al cambiar el valor, éste cambia el valor de su origen. Un control *CheckBox* deshabilitado muestra el valor actual, pero está atenuado y no permite realizar cambios al valor desde la interfaz de usuario.

También puede utilizar casillas de verificación dentro de un cuadro de grupo para seleccionar uno o más elementos relacionados de un grupo. Por ejemplo, puede crear un formulario de pedido que contenga una lista de elementos disponibles, con un control *CheckBox* delante de cada elemento. El usuario puede seleccionar uno o varios elementos determinados activando el control *CheckBox* correspondiente.

La propiedad predeterminada de un control *CheckBox* es *Value* y el evento predeterminado de un control *CheckBox* es *Clic*.

ListBox (Lista): 

Muestra datos o campos en forma de lista, es decir, muestra una lista de valores y le permite seleccionar uno o varios pero no se puede dejar texto dentro del control. Si el

control *ListBox* es dependiente de un origen de datos, almacena el valor seleccionado en el origen de datos.

La propiedad predeterminada de un control *ListBox* es *Value* y el evento predeterminado de un control *ListBox* es *Clic*.

OptionButton (Botón de opción): 

Muestra el estado de selección de un elemento en un grupo de opciones y en base a eso se puede ejecutar una macro o procedimiento.

Utilice un control *OptionButton* para mostrar si está seleccionado un único elemento de un grupo.

Si un control *OptionButton* es dependiente de un origen de datos, puede mostrar el valor de este origen de datos como *Sí*/*No*, *Verdadero/Falso* o *Activado/Desactivado*. Si el usuario selecciona el control *OptionButton*, el valor actual es *Sí*, *Verdadero* o *Activado*; si el usuario no selecciona el control *OptionButton*, el valor es *No*, *Falso* o *Desactivado*. Por ejemplo, un control *OptionButton* en una aplicación de seguimiento de inventario podría mostrar si se ha dejado de distribuir un elemento. Si el control *OptionButton* depende de un origen de datos, al cambiar los valores cambia el valor de este origen de datos.

También puede utilizar un control *OptionButton* dentro de un cuadro de grupo para seleccionar uno o más elementos relacionados de un grupo. Por ejemplo, puede crear un formulario de pedido con una lista de elementos disponibles, con un control *OptionButton* delante de cada elemento. El usuario puede seleccionar un elemento determinado activando el control *OptionButton* correspondiente.

La propiedad predeterminada de un control *OptionButton* es *Value* y el evento predeterminado de un control *OptionButton* es *Clic*.

ComboBox (Caja Combo): 

Combina las características de un control *ListBox* y un control *TextBox*. El usuario puede escribir un valor nuevo, como en un control *TextBox* o bien puede seleccionar un valor existente como en un control *ListBox*.

Si un control *ComboBox* es dependiente de un origen de datos, inserta el valor que el usuario escribe o selecciona en el origen de datos. Si un cuadro combinado de múltiples columnas es dependiente, entonces la propiedad *BoundColumn* determina qué valor se almacena en el origen de datos dependiente.

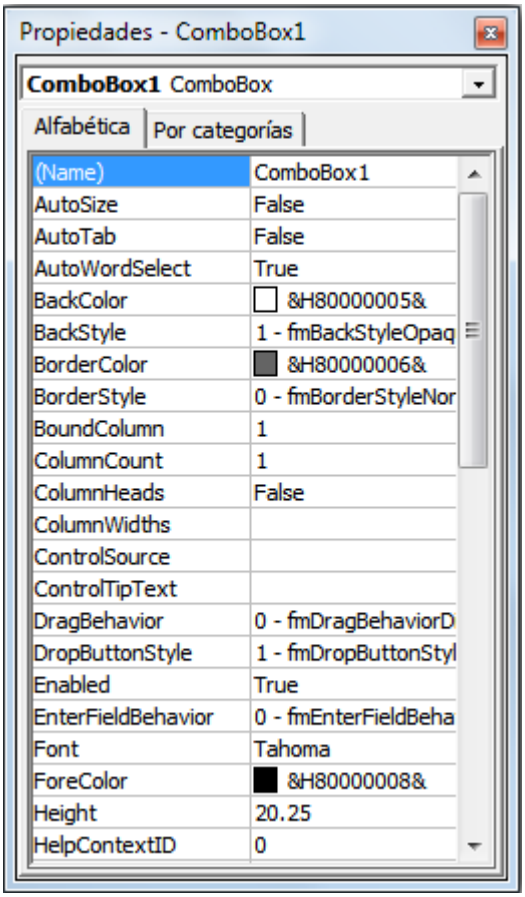
La lista en un control *ComboBox* está formada por filas de datos. Cada fila puede tener una o más columnas, las cuales pueden mostrarse con o sin títulos.

La propiedad predeterminada de un control *ComboBox* es *Value* y el evento predeterminado de un control *ComboBox* es *Change*.

Ventana de propiedades

La ventana de propiedades nos permite personalizar la apariencia y funcionalidad de un formulario o de cualquiera de los controles disponibles en VBA.

La lista de propiedades que muestra la ventana propiedades depende del control que se desea personalizar, aunque hay varias que son comunes a todos (controles) e incluso al *UserForm*.



Haciendo clic en el botón desplegar de la lista ubicada bajo la barra de título de la ventana se muestran todos los controles cargados en el formulario actual. En la ventana de la figura se muestran las propiedades disponibles para un *ComboBox* (cuadro combinado).

Name: Útil para nombrar controles ya que aunque automáticamente se asigna un nombre a cada control introducido (compuesto por de clase de objeto seguido de un entero positivo consecutivo) lo ideal es darle un nombre que lo relacione con el tema sobre el cual estamos haciendo el programa, éste es el nombre con el cual VBA reconoce el control.

AutoSize: Determina si el objeto cambia de tamaño automáticamente para mostrar todo su contenido. El valor *False* mantiene constante el tamaño del control y el texto se recorta cuando excede el tamaño del control y *True* cambia automáticamente el tamaño para que se muestre todo el contenido del control.

AutoWordSelect: Especifica la unidad básica para extender la selección, si el valor es *True* utiliza una palabra como unidad básica de selección, si es *False* utiliza un carácter como unidad básica.

BackColor: Especifica el color de segundo plano del objeto, puede utilizar cualquier número entero que represente un color válido o utilizar la función RGB para establecer el color del objeto. El color de fondo sólo se puede ver si el valor de la propiedad *BackStyle* es *fmBackStyleOpaque*.

Enable: Esta propiedad sirve para habilitar o deshabilitar controles, el valor True habilita el control y este recibe el enfoque y responde a los eventos generados por el usuario, el valor

False lo deshabilita, este se muestra atenuado y no responde a los eventos generados por el usuario pero se puede tener acceso al control mediante código.

EnterFieldBehavior: Especifica el comportamiento de la selección cuando entra en un control *TextBox* o *ComboBox*. El valor *fmEnterFieldBehaviorSelectAll* selecciona todo el contenido del área de edición cuando entra en el control y *fmEnterFieldBehaviorRecallSelection* mantiene sin cambios la selección, visualmente utiliza la selección de la última vez que estuvo activo el control.

Font: Especifica la fuente y el estilo del texto que muestra el control.

ForeColor: Se utiliza para especificar el color de primer plano del control.

Locked: Determina si se puede o no editar un control. El valor *True* no puede editarse el valor, *False* sí puede editarse.

MatchRequired: Especifica si un valor introducido en la parte de texto de un control *ComboBox* debe coincidir con una entrada de la parte de lista existente del control. El usuario puede introducir valores no coincidentes, pero no puede abandonar el control hasta que se introduzca un valor coincidente. Si el valor de *MatchRequeried* es *True* el texto introducido debe coincidir con una entrada existente de la lista, si es False puede ser diferente a la lista existente. Cuando el valor de esta propiedad es *True* el usuario no puede salir del ComboBox hasta que el texto introducido coincida con una entrada existente en la lista, de modo que esta propiedad mantiene la integridad de la lista exigiendo que el usuario elija una entrada existente.

ShowDropButtonWhen: Especifica cuando mostrar el botón de desplegar en un control *ComboBox* o *TextBox*. El valor *fmShowDropButtonWhenNever* no muestra nunca el botón de desplegar, *fmShowDropButtonWhenFocus* sólo cuando tiene el enfoque y *FmShowDropButtonWhenAlways* lo muestra siempre siendo éste el valor predeterminado para el control *ComboBox*; para un *TextBox* el valor predeterminado es *fmShowDropButtonWhenNever*.

Caption: Es el texto descriptivo con el cual se identifica un control. Por ejemplo, al introducir un *CommandButton* el texto que aparece será *CommandButton1* si es el primer botón de comando del formulario, si vamos a programar ese botón para salir del formulario lo ideal sería que mostrara el texto “Cancelar” o “Salir”, esto es lo que nos permite esta propiedad.

Estas son unas de las muchas propiedades con las que, como se mencionó anteriormente, se puede personalizar tanto la apariencia como la funcionalidad de los controles utilizados para crear un programa en VBA.

4) Uso de funciones de Excel en Visual Basic

La mayoría de las funciones de hoja de cálculo de Excel pueden usarse en los enunciados de Visual Basic aunque hay algunas que no son necesarias porque Visual Basic contiene un equivalente de uso fácil, por ejemplo la función concatenar de Excel no es necesaria porque Visual contiene el operador lógico "&" que funciona igual.

Para ejecutar las funciones de hoja de cálculo de Excel en Visual Basic se puede hacer a través del objeto *WorksheetFunction* teniendo presente que para el uso de una función en la que se debe especificar un rango como argumento de la misma se deberá especificar un objeto *Range*, esto es, definir una variable de tipo *Range*.

También puede optarse por introducir una función de hoja de cálculo de Excel desde Visual para lo que deberá especificarse la función como el valor de la propiedad Formula del objeto *Range* correspondiente, por ejemplo:

```
Sub Insertar_Formula()

    Worksheets("Hoja1").Range("A1:C5").Formula = "=RandBetween(0,9)"

End Sub
```

Si copia el código anterior en un módulo al ejecutarlo observe que en el rango de celdas A1:C5 se inserta la función =ALEATORIO.ENTRE(0;9) mostrando su respectivo resultado en cada una de las celdas del rango especificado (números aleatorios entre 0 y 9).

La siguiente es una lista de algunas de las funciones de Excel que pueden ser usadas desde Visual Basic

En Excel	En VBA
Acosh	ACOSH
Amortiz.Progre	AmorDegrc
Amortiz.Lin	Amrlinc
Y	And
Aplicación	Application
Asc	Asc
Aseno	Asin
Asenoh	Asinh
Atan2	Atan2
Promedio	Average
Promedio.Si	AverageIf
Textobaht	BathText
Atanh	Atanh
BesselY	BesselY
Distr.Beta	BetaDist
Limpiar	Clear
Contar	Count
ContarA	CountA
Contar.Blanco	Countblank
Contar.Si	CountIf
Contar.Si.Conjunto	CountIfs
BDPromedio	Daverage
Dias360	Day360
DB	Db
Dbcs	Dbcs
BDContar	Dcount
BDContarA	Dcunta
DDB	Ddb
Dec.A.Bin	Dec2Bin
Dec.A.Hex	Dec2Hex
Dec.A.Oct	Dec2Oct
Grados	Degrees
Delta	Delta
BDProducto	Dproduct
BDSuma	Dsum
Encontrar	Find
Hallar	Search
EncontrarB	FindB
HallarB	SearchB
Frecuencia	Frecuency
EsErr	IsErr
EsError	IsError
EsNoTexto	IsNonText
EsNumero	IsNumber
EsTexto	IsText

En Excel	En VBA
Ln	Ln
Log	Log
Max	Max
Mediana	Median
Min	Min
MInversa	Minverse
MMult	Mmult
Moda	Mode
O	Or
Percentil	Percentile
Pi	Pi
Poisson	Poisson
Precio	Price
Precio.Descuento	PriceDisc
Probabilidad	Prob
Producto	Product
Aleatorio.Entre	Randbetween
Tasa	Rate
Reemplazar	Replace
ReemplazarB	ReplaceB
Redondear	Round
Redondear.Menos	RoundDown
Redondear.Mas	RoundUp
Suma.Series	SeriesSum
SenoH	Sinh
Sustituir	Substitute
Subtotales	Subtotal
Suma	Sum
Sumar.Si	SumIf
Sumar.Si.Conjunto	SumIfs
SumaProducto	SumProduct
Suma.Cuadrados	SumSq
SumaX2MenosY2	SumX2MY2
SumaX2MasY2	SumX2PY2
SumaXMenosY2	SumXMY2
Syd	Syd
TanH	Tanh
Texto	Text
Espacios	Trim
DiaSem	Weekday
Dist.Weibull	Weibull
Frac.Año	YearFrac
Prueba.Z	Ztest
TIRM	MIrr
Repetir	Rept

Los argumentos de las funciones son los mismos tanto para Excel como para Visual Basic. Cualquiera de éstas se debe utilizar anteponiendo la referencia a *WorksheetFunction*

VI. DISEÑO DEL PROGRAMA

Este programa está diseñado para una pequeña o mediana empresa que por alguna razón no posea un sistema informatizado para el registro y control de sus operaciones de modo que a través de él pueda llevar el control de su planilla de pagos, siempre en Excel pero con las ventajas que se mencionaron al inicio del documento.

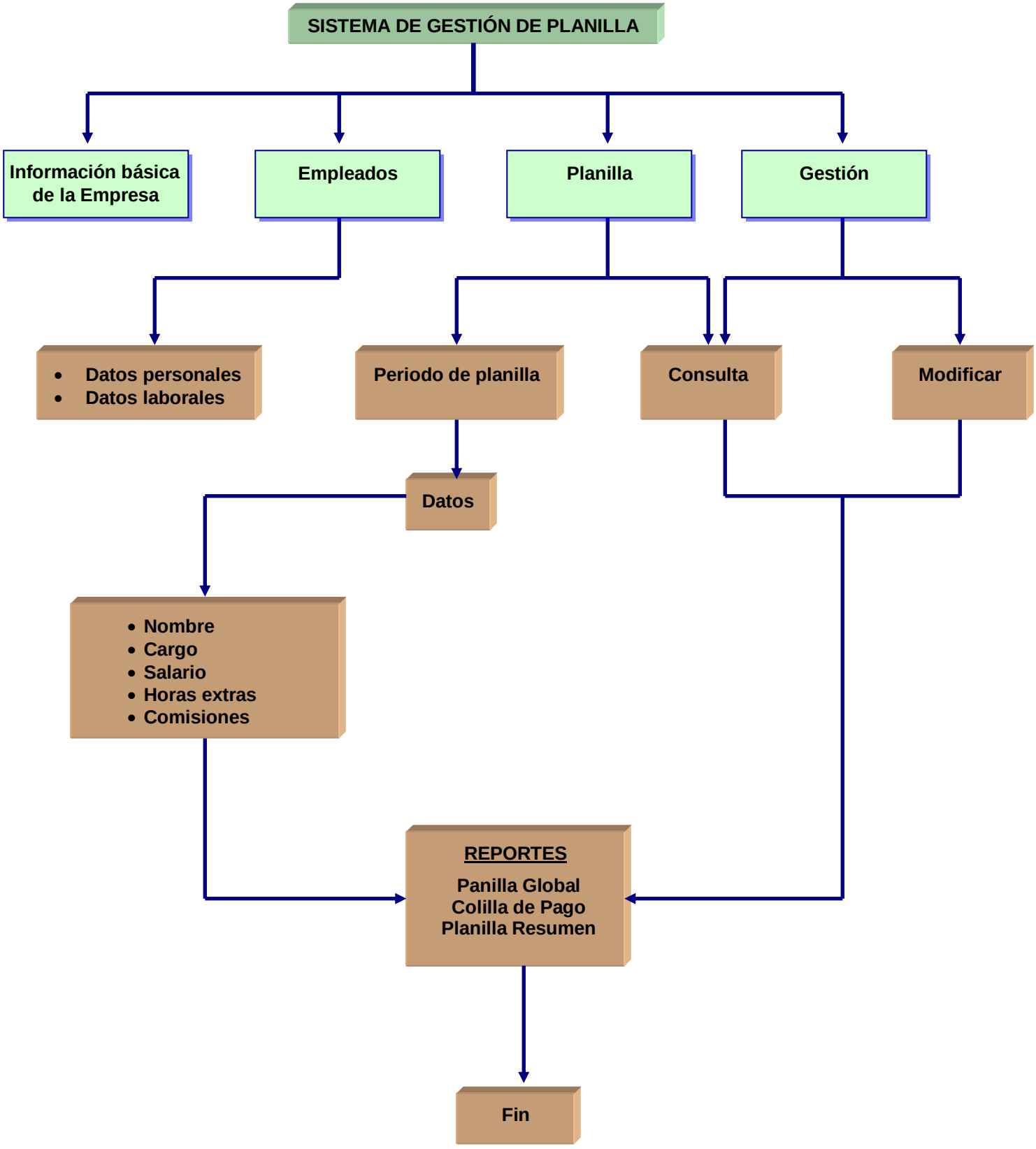
La información necesaria para preparar la planilla será en todo caso la misma que se esté usando para su elaboración sea cual fuera el medio o herramienta utilizada hasta el momento, se comprenderá que esto se debe a que existe una reglamentación para dichos cálculos.

Los datos indispensables para realizar el cálculo de la planilla de pagos son los datos generales del trabajador, el salario mensual y la fecha de ingreso del trabajador. Comisiones, horas extras y otras bonificaciones al igual que otras deducciones es información complementaria y que no siempre habrá que ingresar.

Para agilizar la elaboración de la nómina se ha diseñado un módulo en el cual se debe ingresar la información mínima requerida de modo que el programa no de error o más bien se pueda ejecutar.

PANTALLA DE INICIO

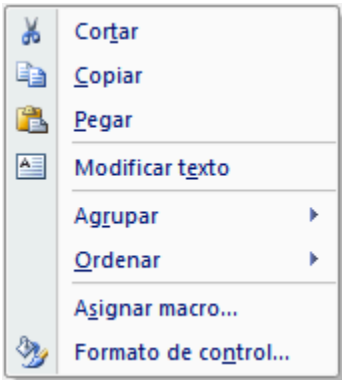
Algoritmo del programa



La primera vez que se vaya a usar el programa habrá que ingresar la información básica de la empresa esto es, nombre completo, nombre corto o siglas, número RUC, dirección y el logotipo; ésta es la información que aparecerá en cada uno de los reportes que se generen usando el programa. Para entrar esta información se utiliza el módulo INFORMACIÓN BÁSICA DE LA EMPRESA.

De igual manera habrá que introducir los datos mínimos de los empleados requeridos por el programa para la elaboración de la nómina, esto se hace a través del módulo EMPLEADOS, en el módulo CREAR NÓMINA es donde se elabora en sí la planilla de pagos mientras que, el módulo GESTIÓN se utiliza para realizar consultas, generar la colilla de pago de cada uno de los trabajadores, generar una versión para imprimir de la planilla, generar un reporte de totales a partir de la planilla e importar la planilla a un libro de Excel; el que puede ser usado para realizar otros cálculos que la dirección de la empresa considere necesarios.

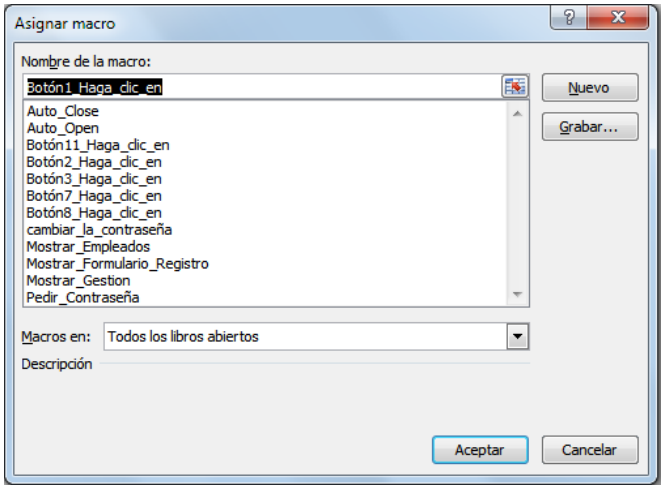
Como se mencionó anteriormente, para ingresar los datos al programa no se utiliza la hoja de cálculo sino que hemos diseñado y programado formularios en Visual Basic para Aplicaciones para tal propósito, así que para llamar o activar dichos formularios sobre la hoja de cálculo de Excel hemos usado los controles de formularios que posee Excel.



Haciendo clic en la ficha programador y luego en el botón insertar, en el panel control de formulario dar clic sobre el control "Botón" para luego dibujar un botón en cualquier parte de la hoja de cálculo, dando clic derecho sobre el botón dibujado, éste despliega una serie de opciones que permiten personalizar su apariencia y asignarle funciones.

La opción modificar texto permite cambiar el texto que por defecto aparece sobre el control, que es el indicador del tipo seguido de un número consecutivo, formato de control sirve para dar formato al control y la más importante para nuestro trabajo, Asignar macro, asigna una macro al control la que se ejecutará cuando se de clic sobre el control.

Haciendo clic sobre la opción Asignar macro se despliega una ventana que contiene todas las macros disponibles, en nuestro caso las que se muestran a continuación:



Seleccionamos la macro que queremos asignar al Botón y damos clic en Aceptar.
Si bien es cierto que en nuestro caso únicamente asignamos una macro que carga y muestra un formulario a un botón control de formulario se puede asignar cualquier macro sin importar la complejidad del código de la macro.

Una de las macro asignada a los botones utilizados es la siguiente:

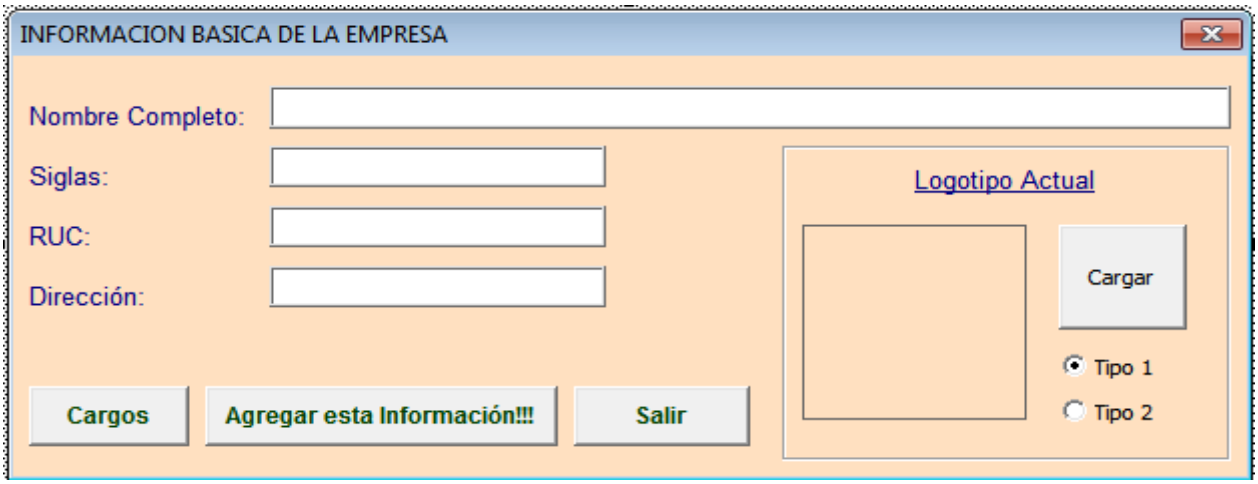
```
Sub Mostrar_Empleados()
    Load Empleados
    Empleados.Show
End Sub
```

Sub y *End Sub* indican el inicio y fin del procedimiento, *Mostrar_Empleados* es el nombre del procedimiento. Hemos diseñado un formulario para introducir los datos de los empleados a la base de datos al que llamamos *Empleados*, así que esta macro lo que hace es cargar (*Load*) y mostrar (*Show*) el formulario. Esta misma estructura poseen todas las macros asignadas a cada uno de los botones que se muestran en la pantalla de inicio del programa.

1) Formularios en Visual Basic

Un formulario (*UserForm*) no es más que una ventana en la que se pueden dibujar controles que luego se deberán programar para dar vida a un programa.

Usando los controles *TextBox*, *Frame*, *Label*, *CommandButton*, *Image* y *OptionButton* hemos diseñado el formulario a través del cual se introduce la información básica de la empresa.



Los botones de opción Tipo 1 y Tipo 2 permiten seleccionar una imagen de mayor o menor resolución (el de mayor resolución hace que el programa gane peso), después de digitar la información en los cuadros de texto y hacer clic en el botón Agregar esta información

habremos cambiado o modificado los datos básicos de la empresa que se muestran en cada uno de los reportes que se generan.

Como mencionamos anteriormente, un botón de comando sirve para encapsular un procedimiento que se ejecuta haciendo clic sobre el; esto es lo que hace el botón Agregar esta información de la siguiente manera.

```
Private Sub B_Agregar_Clic()
    Dim Mensaje As String

  'Definimos el mensaje a mostrar al  cambiar la información

  Mensaje = "Nombre cambiado con éxito!!!, Nuevo nombre: " &
  T_Nombre.Value & ", Abreviatura: " & T_Siglas.Value

'Guardamos el nombre, siglas, RUC y dirección en el rango U10:U13 de hoja
inicio

     Range("U10").Value = T_Nombre.Value Z
     Range("U11").Value = T_Siglas.Value
     Range("U12").Value = T_RUC.Value
     Range("U13").Value = T_Direccion.Value

'Ubicamos la imagen en el control imagen de las hojas 1, 3 y 4

    Hoja1.Image1.Picture = Image1.Picture
    Hoja1.Image1.PictureSizeMode = Image1.PictureSizeMode
    Hoja3.Image1.Picture = Image1.Picture
    Hoja3.Image1.PictureSizeMode = Image1.PictureSizeMode
    Hoja3.Range("F4").Value = T_Direccion.Value
    Hoja3.Range("F5").Value = T_RUC.Value
    Hoja4.Image1.Picture = Image1.Picture
    Hoja4.Image1.PictureSizeMode = Image1.PictureSizeMode

'El nombre, siglas, RUC y dirección a las hojas 4, 9 y 31

    Hoja4.Range("B2").Value = T_Nombre
    Hoja4.Range("B3").Value = T_Siglas
    Hoja9.Range("A1").Value = T_Nombre.Value
    Hoja9.Range("G2").Value = T_Siglas.Value
    Hoja9.Range("J2").Value = "RUC: " & T_RUC.Value
    Hoja9.Range("J3").Value = T_Direccion.Value
    Hoja31.Image1.Picture = Image1.Picture
    Hoja31.Image1.PictureSizeMode = Image1.PictureSizeMode
    Hoja31.Range("F3").Value = "RUC: " & T_RUC.Value
    Hoja31.Range("A4").Value = T_Direccion.Value

'Mostrar el nombre y las siglas en la pantalla de inicio

    Hoja1.Label1.Caption = Range("U10").Value
    Hoja1.Label2.Caption = Range("U11").Value
```

```
'Mostramos el mensaje

    MsgBox    Prompt:=Mensaje,    Buttons:=vbInformation,    Title:="Nombre
Cambiado!!!"

    Unload EMPRESA

End Sub
```

Con esto habremos cambiado o agregado la información básica de la empresa, la que quedará guardada en la pantalla de inicio y en todos los informes que posteriormente se generarán.

El procedimiento funciona de la siguiente manera:

A la variable *Mensaje* se le asigna el mensaje que se muestra cuando damos clic sobre el botón Agregar esta información usando un *MsgBox*, los *TextBox* llamados *T_Nombre*, *T_Siglas*, *T_RUC* y *T_Direccion* es donde el usuario introduce el nombre, las siglas, el número RUC y la dirección de la empresa; con el control *Image* cargamos la imagen logotipo de la empresa, luego utilizando la propiedad *Value* del objeto *Range* guardamos estos datos en las hojas de cálculo donde se necesita.

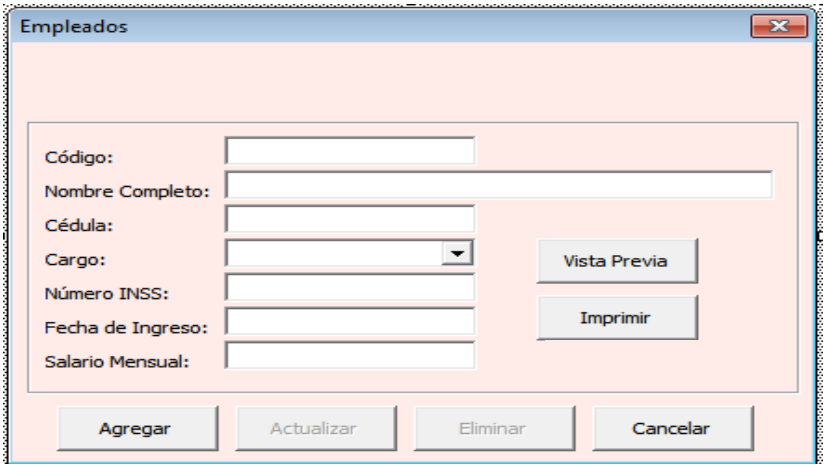
El botón *Cargo* muestra un formulario en el que se pueden agregar todos los posibles cargos a los que puede aspirar u ocupar un trabajador en la empresa, de modo que el código que se ejecuta al hacer clic sobre ese botón es:

```
Private Sub B_Cargos_Clic()
    Load Cargo
    Cargo.Show
End Sub
```

La cláusula *Private* permite ahorrar memoria, lo que hace que el programa corra con mayor rapidez, aunque tiene el inconveniente que el procedimiento no puede ser llamado desde otro.

2) Módulo Empleados

Al igual que el módulo de INFORMACIÓN BÁSICA DE LA EMPRESA el módulo EMPLEADOS, esta asignado a un control de formulario que ejecuta la macro que carga y muestra el formulario desde donde se digitará la información básica de cada uno de los empleados de la empresa.



El código del empleado es identificador o clave que se usa a lo largo del proceso de registro, creación de la planilla, consultas y generación de informes por lo que debemos asegurarnos que éste sea único para cada empleado. Con el código de programación mostrado a continuación controlamos eso.

```
Private Sub T_Codigo_Change()

'Se definen las varialbles
'(Variables)

'Activamos la hoja donde se guarda la base de datos de los empleados
    Hoja31.Activate
'Activamos la primera celda donde se guardan los registros
    Hoja31.Range("B7").Activate

    Do
'Si hay un valor en la celda activa asignamos el código a la variable
código

    If Not IsEmpty(ActiveCell) Then
        Codigo = T_Codigo.Value

'Si el código escrito ya existe se copian los datos del registro en el
formulario

        If ActiveCell.Value = Codigo Then
            Posicion = ActiveCell.Offset(0, -1).Value
            Nombre = ActiveCell.Offset(0, 1).Value
            Cedula = ActiveCell.Offset(0, 2).Value
            Cargo = ActiveCell.Offset(0, 3).Value
            INSS = ActiveCell.Offset(0, 4).Value
            Ingreso = ActiveCell.Offset(0, 5).Value
            Salario = ActiveCell.Offset(0, 6).Value
            T_Nombre.Value = Nombre
            T_Cedula.Value = Cedula
            C_Cargo.Value = Cargo
            T_INSS = INSS
```

```
            T_Ingreso = Ingreso
            T_Salario = WorksheetFunction.Text(Salario, "#,##0.00")
            T_Posicion.Value = Posicion

'Inhabilitamos el botón Agregar y habilitamos Actualizar y Eliminar

            B_Agregar.Enabled = False
            B_Actualizar.Enabled = True
            B_Eliminar.Enabled = True
            Encontrado = True

'Si el código no se ha asignado habilitamos el botón agregar e
inhabilitamos los otros

        Else

            B_Agregar.Enabled = True
            B_Actualizar.Enabled = False
            B_Eliminar.Enabled = False
            ActiveCell.Offset(1, 0).Activate
            Encontrado = False

        End If

    Else

        T_Nombre.Value = Empty
        T_Cedula.Value = Empty
        C_Cargo.Value = Empty
        T_INSS = Empty
        T_Ingreso = WorksheetFunction.Text(Now(), "dd/mm/yyyy")
        T_Salario = Empty

        Exit Sub
    End If

    Loop Until Encontrado = True
  Application.ScreenUpdating = True
End Sub
```

El procedimiento anterior lo que hace es verificar si el código que asignamos al empleado ya ha sido asignado a otro, si es así carga los datos al formulario y habilita los botones Actualizar y Eliminar con los que se puede actualizar la información del empleado o eliminarlo definitivamente de la base de datos; si no existe coincidencia entre la clave de empleado digitada y las existentes en la base el programa permite que se introduzcan los datos y se guarde el registro.

Una vez se haya llenado el formulario con todos los datos requeridos necesitamos guardar esa información en nuestra base de datos, esto lo hace la macro asignada al botón Agregar.

```
Private Sub B_Agregar_Clic()
  'Se definen las variables
(Variables)

    On Error GoTo Datos_empleados

    Hoja31.Activate
    Hoja31.Range("B7").Activate

'Buscar la primera celda vacía

    Do While Not IsEmpty(ActiveCell)
        ActiveCell.Offset(1, 0).Activate
    Loop

    Do
'Asignamos los datos a las variables
        Codigo = T_Codigo.Value
        Nombre = T_Nombre.Value
        Cedula = T_Cedula.Value
        Cargo = C_Cargo.Value
        INSS = T_INSS
        Ingreso = DateValue(T_Ingreso.Value)
        Salario = T_Salario

'Guardamos los datos en su respectiva ubicación
    With ActiveCell
            .Value = Codigo
            .Offset(0, 1).Value = Nombre
            .Offset(0, 2).Value = Cedula
            .Offset(0, 3).Value = Cargo
            .Offset(0, 4).Value = INSS
            .Offset(0, 5).Value = Ingreso
            .Offset(0, 6).Value = Salario
    End With
'Mostramos un mensaje al usuario cuando ya se ha guardado el registro
        Mensaje = "Empleado " & Nombre & " ha sido agregado
exitosamente!"
        MsgBox Prompt:=Mensaje, Buttons:=vbInformation,
Title:="Empleados"

'Limpiamos los TextBox para ingresar otro registro
        T_Codigo.Value = Empty
        T_Nombre.Value = Empty
        T_Cedula.Value = Empty
        C_Cargo.Value = Empty
        T_INSS = Empty
        T_Ingreso = Now()
        T_Salario = Empty

        Exit Sub
```

```
    Loop While T_Codigo = ""

Datos_empleados:
    MsgBox Prompt:="Introduzca datos requeridos", Buttons:=vbInformation,
Title:="Error!"

    Exit Sub

End Sub
```

Al guardar un nuevo registro lo primero que tenemos que hacer es ir a la hoja donde se guardan los registros y encontrar la primera celda vacía, una vez asignados los valores a cada una de las variables procedemos a guardarlas en su respectiva celda, si no se introducen datos en alguno de los cuadros de texto (*TextBox*) o los tipos no coinciden el programa genera un error esto lo tratamos con la rutina *On Error GoTo Datos_empleados* así, si ocurre un error el programa se dirige a ejecutar el código de la etiqueta *Datos_empleados* la que informa del error y finaliza el procedimiento.

El botón "Vista Previa" activa la vista previa propia de Excel de modo que el usuario puede observar la forma en que quedarían impresos los datos o mejor dicho la base de datos de los empleados.

```
Application.ScreenUpdating = False
    Hoja31.Range("B7").Activate

    Do While Not IsEmpty(ActiveCell)
        ActiveCell.Offset(1, 0).Activate
        Filas = ActiveCell.Offset(0, -1).Value + 5
    Loop

    Rango_Datos = "$A$1:$H$" & Filas
    ActiveSheet.PageSetup.PrintArea = Rango_Datos
    Empleados.Hide
    Hoja31.Visible = xlSheetVisible

Application.ScreenUpdating = True

    ActiveWindow.SelectedSheets.PrintPreview
    ActiveSheet.PageSetup.PrintArea = ""
```

La hoja 31 del libro de trabajo es la que se usa para guardar la base de datos de los empleados, así que usando el método *Activate* activamos la celda "B7" que contiene el primer registro, luego buscamos la última fila que contiene datos para establecer el área de impresión de modo que en la vista previa se muestren únicamente las páginas que contienen datos. Para hacer más amigable el entorno del programa cuando se ejecuta este código se oculta el formulario "Empleados" y establecemos en *False* la propiedad *ScreenUpdating* del objeto *Application* (*ScreenUpdating* controla las actualizaciones de pantalla, estando en valor *False* la pantalla no se actualiza), una vez se muestra la vista previa se borra el área

de impresión para que sea establecido nuevamente hasta que se vuelva a ejecutar el procedimiento.

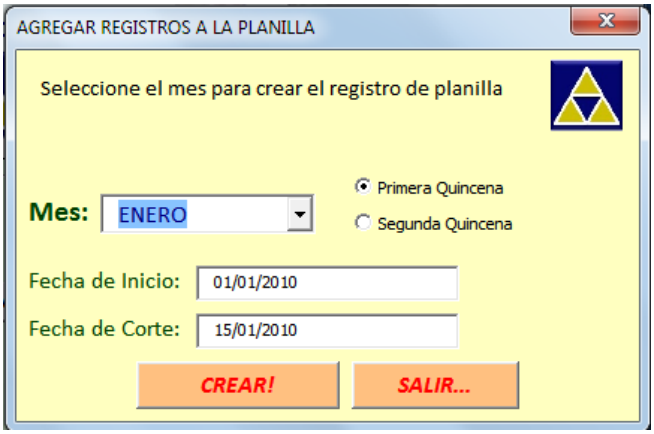
Cuando se oprime el botón "Imprimir" se ejecuta un procedimiento similar en código al que muestra la vista previa, esto es; activa la hoja que contiene la base de datos, busca la última fila que contiene datos para establecer el área de impresión y luego la manda a la impresora predeterminada.

`ActiveSheet.PrintOut` es la línea de código que manda a imprimir el área de impresión definido en la hoja activa del libro de trabajo, que de hecho es donde se encuentra la base de datos de los empleados.

3) Módulo Nómina

Podríamos afirmar que este es el módulo más interesante, ya que es aquí donde realmente se elabora la nómina o planilla de pagos.

Inicialmente, al hacer clic sobre botón (control de formulario) CREAR NOMINA se activa el formulario que se muestra a continuación, donde se debe seleccionar el mes y la quincena a pagar.



En el *ComboBox* llamado *C_Meses* (por comodidad el nombre de todos los controles usados en el programa esta formado por el indicador del tipo de control del que se trata, por ejemplo C para un *ComboBox*, T para un *TextBox*; seguido de un guión bajo y una palabra relacionada con el contenido o acción que se ejecuta a través de el) de la lista desplegable se puede seleccionar el mes al que corresponde la planilla (también lo puede digitar desde el teclado), con los botones de opción podemos seleccionar la quincena y en los cuadros de texto "Fecha de Inicio" y "Fecha de Corte" escribimos la fecha exacta para la que se prepara la planilla de pagos. Asignando al evento *Initialize* del *UserForm Crear_Registro* el siguiente código hacemos que se muestren los doce meses del año.

```
With Crear_Registro.C_Meses
     .AddItem "ENERO"
     .AddItem "FEBRERO"
     .AddItem "MARZO"
     .AddItem "ABRIL"
     .AddItem "MAYO"
     .AddItem "JUNIO"
     .AddItem "JULIO"
     .AddItem "AGOSTO"
     .AddItem "SEPTIEMBRE"
     .AddItem "OCTUBRE"
     .AddItem "NOVIEMBRE"
     .AddItem "DICIEMBRE"
End With
```

El método *AddItem* permite agregar un artículo, en este caso los meses, al cuadro de lista combinado.

Se ha programado para que por defecto se muestre el mes actual y se seleccione la primera quincena de pagos, esto lo hacemos así:

```
C_Meses.ListIndex = Month(Now()) - 1
```

La propiedad *ListIndex* identifica el elemento seleccionado actualmente ya sea en un control *ListBox* o *ComboBox*. Esta propiedad contiene un índice de la fila seleccionada en una lista donde el intervalo de valores es desde -1 a uno menos que el número total de filas de una lista es decir, *ListCount – 1*, por eso al resultado de la fusión de las funciones *Month* y *Now* (*Now* devuelve la fecha de hoy en formato día, mes, año, hora y minutos y *Month* devuelve el mes de una fecha cualquiera) le restamos 1.

Siempre dentro del mismo procedimiento hacemos que se seleccione la quincena que corresponde a la fecha actual, es decir, si entramos al módulo CREAR REGISTRO entre el día 1 y 15 de determinado mes estará activo el botón de opción “Primera Quincena” y en los cuadros *Fecha de Inicio* y *Fecha de Corte* aparecerá 01/01/2010 y 15/01/2010 respectivamente suponiendo que estamos en Enero de 2010.

```
If Day(Now()) <= 15 Then

     O_PrimeraQuincena.Value = True
     T_Inicio = DateSerial(Year(Now), C_Meses.ListIndex + 1, 1)
     T_Corte = DateSerial(Year(Now), C_Meses.ListIndex + 1, 15)

Else

     O_SegundaQuincena.Value = True
     T_Inicio = DateSerial(Year(Now), C_Meses.ListIndex + 1, 16)
     T_Corte = DateSerial(Year(Now), C_Meses.ListIndex + 1, 30)

End If
```

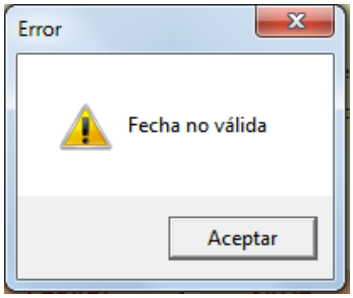
El valor *True* activa un *OptionButton*, lo que sucede cuando el día de hoy es 15 o menos y luego en el cuadro de texto se escriben las fechas de inicio (*T_Inicio*) y corte (*T_Corte*) de pago usando la función *DateSerial* (*DateSerial* es una función de Visual Basic y no pertenece al objeto *Worksheetfunction*) que tiene formato de fecha año, mes, día. Con la estructura condicional *If Else* controlamos los dos posibles escenarios que se pueden presentar: Primera quincena o Segunda quincena aunque para los meses de treinta y un días hay que cambiar manualmente la fecha de corte.

Ya que en la medida de lo posible tratamos de prever los escenarios que se puedan presentar con respecto a las fechas también tomamos en cuenta el caso de febrero que tiene veintiocho días aunque también hay que cambiar de forma manual cuando el año sea bisiesto.

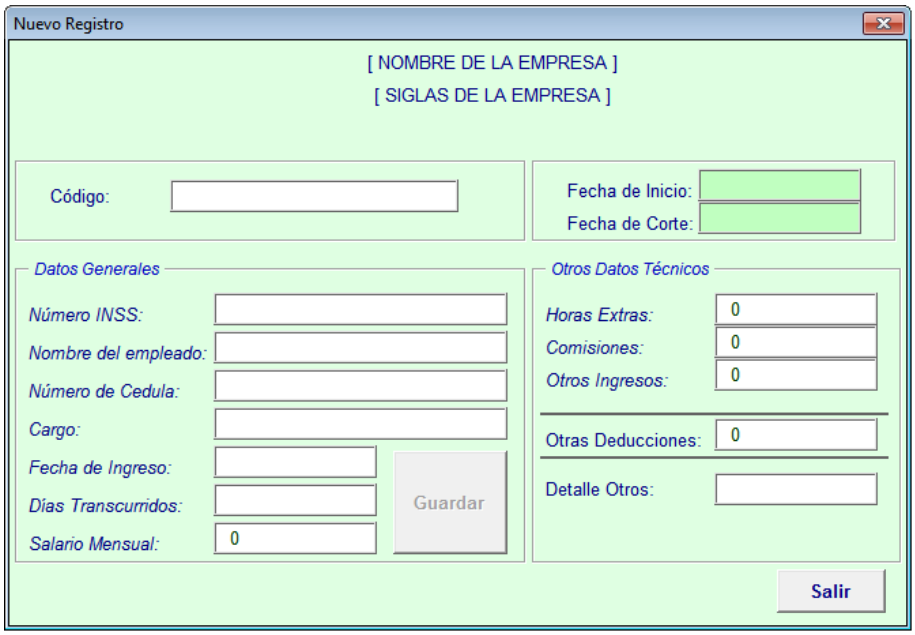
```
If C_Meses.ListIndex = 1 Then

        T_Inicio = DateSerial(Year(Now), C_Meses.ListIndex + 1, 16)
        T_Corte = DateSerial(Year(Now), C_Meses.ListIndex + 1, 28)
```

Así, si el número de la lista de los meses en *C_Meses* es 1 (corresponde a febrero porque *ListIndex* cuenta desde cero) entonces la fecha de corte es el 28/02/2010 (asumiendo que estamos en el año 2010). Esta forma de contar de *ListIndex* explica porque sumamos uno al valor que da *C_Meses.ListIndex*.

Nota: De no ingresar de forma correcta la fecha para el cálculo se produciría un error en tiempo de ejecución y se mostraría el siguiente mensaje, teniendo que corregir la fecha.



Una vez hayamos seleccionado la fecha correcta y dado clic en el botón CREAR se muestra el siguiente formulario, en el que se deberán ingresar los datos requeridos tanto para el cálculo como para la correcta presentación de la planilla de pagos.



El código del empleado es un identificador único que se ha agregado a todos y cada uno de los empleados registrados en la base de datos de empleados de la empresa. Estando en esta pantalla con sólo digitar el código del empleado en el cuadro de texto correspondiente automáticamente se rellenan los demás cuadros con los datos que con anterioridad había sido registrado el empleado.

```
Do
    If Not IsEmpty(ActiveCell) Then
        Codigo = T_Codigo.Value
        If ActiveCell.Value = Codigo Then
            Posicion = ActiveCell.Offset(0, -1).Value
            Nombre = ActiveCell.Offset(0, 1).Value
            Cedula = ActiveCell.Offset(0, 2).Value
            Cargo = ActiveCell.Offset(0, 3).Value
            INSS = ActiveCell.Offset(0, 4).Value
            Ingreso = ActiveCell.Offset(0, 5).Value
            Salario = ActiveCell.Offset(0, 6).Value
            Corte = T_Corte.Value
            T_Nombre.Value = Nombre
            T_Cedula.Value = Cedula
            T_Cargo.Value = Cargo
            T_INSS = INSS
            T_Ingreso = Ingreso
            T_Salario = WorksheetFunction.Text(Salario, "#,##0.00")
            T_Posicion.Value = Posicion
            T_Dias = Corte - Ingreso
            B_GuardarRegistro.Enabled = True
            Encontrado = True
        Else
            ActiveCell.Offset(1, 0).Activate
            B_GuardarRegistro.Enabled = False
            Encontrado = False
        End If
    Else
```

```
        T_Nombre.Value = Empty
        T_Cedula.Value = Empty
        T_Cargo.Value = Empty
        T_INSS = Empty
        T_Ingreso = Empty
        T_OtrasDeduc = WorksheetFunction.Text(0, "##,##0.00")
        T_OtrosIng = WorksheetFunction.Text(0, "##,##0.00")
        T_Comisiones = WorksheetFunction.Text(0, "##,##0.00")
        T_Detalle = Empty
        T_Dias = Empty
        T_Salario = WorksheetFunction.Text(0, "##,##0.00")
        Exit Sub
    End If
    Loop Until Encontrado = True
```

Aquí usamos la función texto (*WorksheetFunction.Text*) para que al rellenarse los cuadros de texto con datos numéricos éstos se muestren con el formato acostumbrado de número (coma como separador de miles, punto como separador de decimales y con dos decimales).

Como el código anterior esta asignado al evento *Change* del control *TextBox* donde se debe escribir el código del empleado, éste (bloque de código) se ejecutará con cada cambio que se haga y por tanto el contenido de los demás cuadros de texto se debe actualizar, si no existe registro (empleado) que tenga asignado el código los cuadros de texto donde debería mostrarse la información permanecerán vacíos, esto es lo que hace el bucle del último *Else* donde asignamos el valor *Empty* para cada uno de los cuadros de texto (*Empty* indica que una variable *Variant* no tiene asignado ningún valor inicial. Las variables *Empty* son iguales a 0 (en un contexto numérico) o de longitud cero ("") (en un contexto de cadena), en un control *TextBox* borra el contenido del mismo).

Una vez ingresada y verificada la información del empleado el siguiente paso es efectuar los cálculos correspondientes y de paso mandar el resultado a la hoja de cálculo donde se están guardando los datos correspondientes a la quincena para la cual se prepara la planilla de pagos.

Dando clic sobre el botón Guardar se ejecuta el código que primeramente hace los cálculos de asignaciones y deducciones para obtener el salario a recibir por parte del trabajador y luego manda el resultado de dichos cálculos a la hoja que corresponde el periodo de pago. Como desde el momento en que entramos a la pantalla "Nuevo Registro" activamos la hoja donde se guardarán los datos y luego el procedimiento que verifica que el registro aún no ha sido agregado nos deja en la primera celda vacía, usando la instrucción *With* y la propiedad Offset procedemos a guardar los datos.

```
With ActiveCell

        .Value = Codigo
        .Offset(0, 1).Value = Nombre
        .Offset(0, 2).Value = Salario
        .Offset(0, 3).Value = Antiguedad
        .Offset(0, 4).Value = Extras_Cordobas
        .Offset(0, 5).Value = Comisiones
        .Offset(0, 6).Value = OtrosIng
```

```
        .Offset(0, 7).Value = Devengado
        .Offset(0, 8).Value = INSS
        .Offset(0, 9).Value = IR
        .Offset(0, 10).Value = OtrasDeduc
        .Offset(0, 11).Value = Deducciones
        .Offset(0, 12).Value = Neto
        .Offset(0, 13).Value = Patronal
        .Offset(0, 14).Value = Extras
        .Offset(0, 15).Value = Dias
        .Offset(0, 16).Value = Numero_INSS
        .Offset(0, 17).Value = Ingreso
        .Offset(0, 18).Value = T_Cedula
        .Offset(0, 19).Value = Detalles
        .Offset(0, 20).Value = Cargo
        .Offset(0, 21).Value = Rango

End With
    Mensaje = "El registro: " & Nombre & ", se ha agregado
exitosamente!!!"
    Mas_Datos = MsgBox(Mensaje, vbInformation, "NOMINA")
```

Al final se muestra un mensaje indicándonos que el registro ha sido agregado correctamente.

Como medida de seguridad debemos asegurarnos que el registro que deseamos crear aún no existe. El código que se muestra a continuación busca en cada una de las celdas no vacías una que coincida con el código del empleado al que se desea hacer el cálculo, si lo encuentra se muestra el mensaje siguiente y el programa se sale del procedimiento.


```
Do
    Codigo = T_Codigo.Value
    If Codigo = ActiveCell.Value Then
        MsgBox Prompt:= "Este Registro ya existe!",
Buttons:=vbExclamation, Title:="Registro"
        T_Codigo.SetFocus
        Exit Sub
```

```
    Else
        ActiveCell.Offset(1, 0).Activate
    End If
Loop While Not IsEmpty(ActiveCell)
```

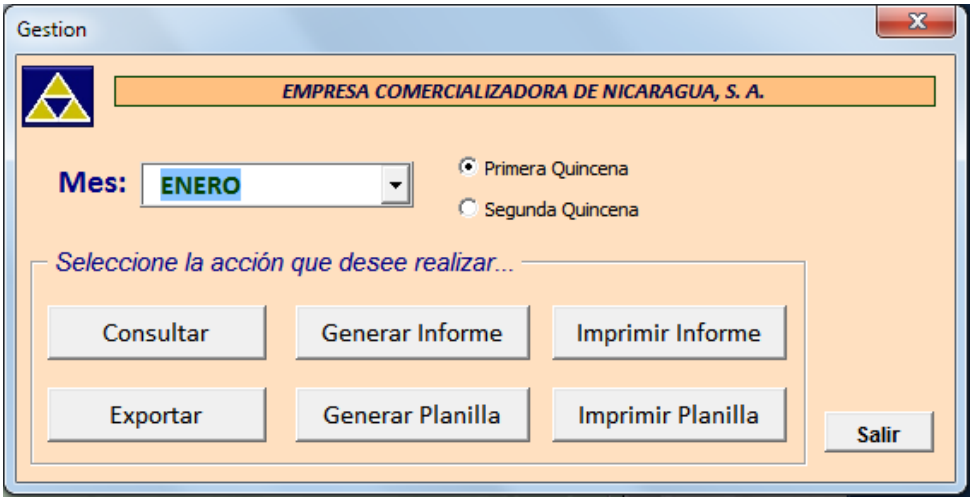
Eso es lo que hace la primera parte de la estructura condicional. Si el valor de la celda activa no coincide con el valor de la variable código entonces se activa la siguiente celda hacia abajo y se comprueba nuevamente el valor de esa celda; esto nos asegura que de no encontrarse el registro esta parte del procedimiento nos deja en la primera celda vacía, que es donde vamos a guardar los datos correspondientes a cada nuevo registro.

Si el registro no se encuentra entonces se ejecuta el resto del procedimiento que consiste en realizar los cálculos correspondientes a cuota laboral y patronal del INSS, el impuesto sobre la renta, sumar o restar otras asignaciones y deducciones para obtener el salario neto a recibir por el trabajador, guardar el registro, volver al formulario de entrada de datos y borrar los datos del empleado anterior para ingresar otro.

4) Módulo de Gestión

Una vez que hayamos realizado el cálculo para todos y cada uno de los empleados el siguiente paso sería generar la planilla, la colilla de pago y el informe que muestra el consolidado en concepto de sueldos y salarios, deducciones y neto a pagar.

Para esto hemos creado el módulo GESTIÓN en el que además podemos tener una vista previa de los informes antes mencionados y adicionalmente podemos exportar la planilla a un nuevo libro de Excel en el que podamos realizar los cálculos que por alguna razón sean necesarios por la administración.

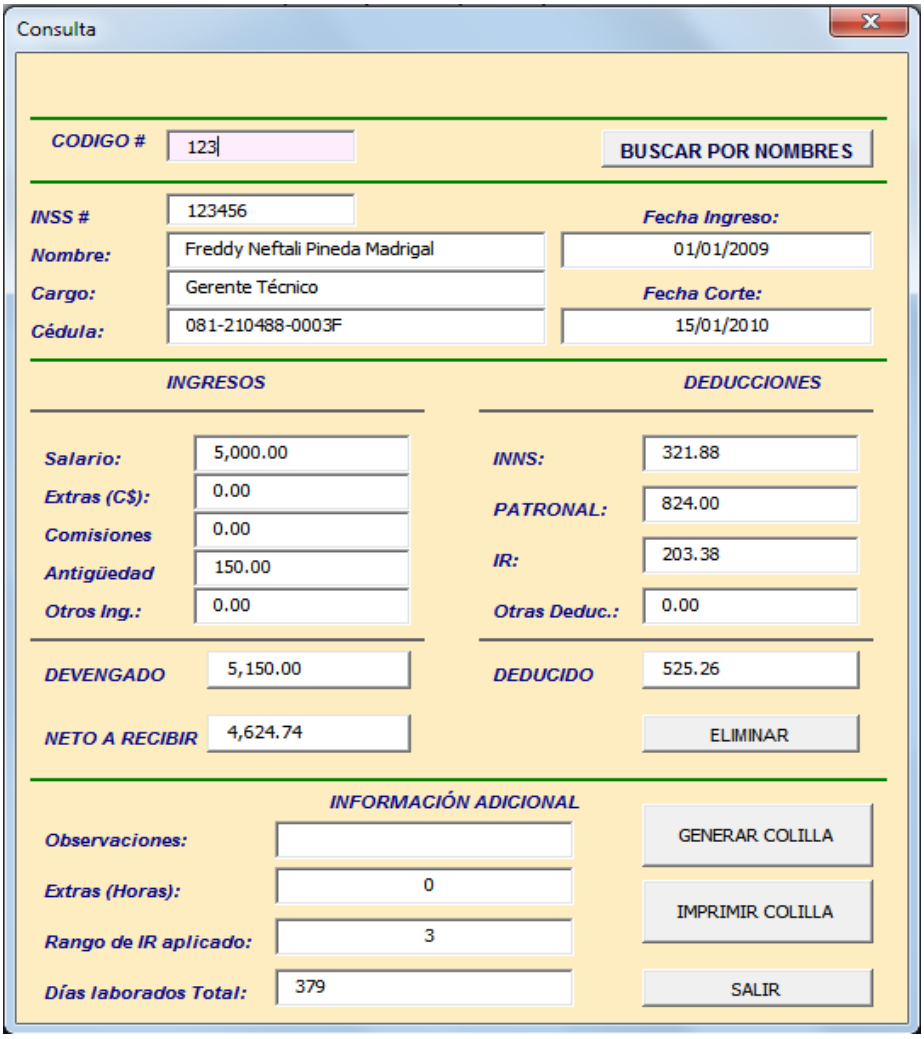


Una vez estando en el módulo de Gestión tenemos seis opciones:

1. Consultar
2. Exportar
3. Generar informe
4. Generar planilla
5. Imprimir informe
6. Imprimir planilla

1. Consultar

El botón Consultar despliega el formulario "Consulta" en el que, como su nombre lo indica, podemos realizar una consulta en la base de datos correspondiente al mes y quincena seleccionados en Gestión.



Aquí hay dos opciones para hacer una consulta. Una es ingresar el código del empleado en el cuadro de texto CODIGO #, el que tiene asignado al evento *Change* un bloque de

códigos que busca automáticamente (sin hacer clic en ningún botón) los datos correspondientes al empleado cuyo código se digitó.

Con el código (bloque de sentencias) asignado al evento *Change* del *TextBox* CODIGO buscamos en la hoja donde están los datos, en la columna correspondiente al código del empleado el que sea exactamente igual al digitado en el cuadro de texto y asignamos los datos a las variables para luego asignarlo a los correspondientes cuadros del texto del formulario Consulta.

Una segunda opción es buscar por nombre, dando clic sobre el botón BUSCAR POR NOMBRES el cuadro de texto NOMBRE se convierte en un cuadro de lista combinado (*ComboBox*) que muestra todos los registros existen en la base de datos en la que se hace la consulta.

Para agregar todos los nombres a la lista del *ComboBox* utilizamos la estructura repetitiva *For* de la siguiente manera (Este procedimiento esta asignado al evento clic del botón buscar por nombres).

```
For i = 6 To 500

Cel = "C" & i
If Range(Cel).Value <> "" Then
   C_Nombre.AddItem Range(Cel).Value
End If
Next i
```

La lógica para buscar por nombres es la misma que la utilizada para buscar por código de empleado con la única diferencia que lo que debe coincidir es el nombre por lo que el procedimiento utilizado para ubicar el registro busca en la columna correspondiente a los nombres.

Una vez que hayamos (por cualquiera de los dos métodos) ubicado los datos del empleado que buscamos tenemos la opción de eliminar definitivamente el registro (esta es la única forma de hacer correcciones cuando se cometen errores al realizar los cálculos de la planilla de pagos) o generar la colilla de pagos para después poder imprimirla.

Eliminar registros

En la pantalla de CONSULTA se muestra el resultado de los cálculos realizados en el módulo CREAR NOMINA, si por alguna razón se cometió un error tenemos que eliminar definitivamente el registro y luego ir nuevamente a CREAR NOMINA para generarlo nuevamente.

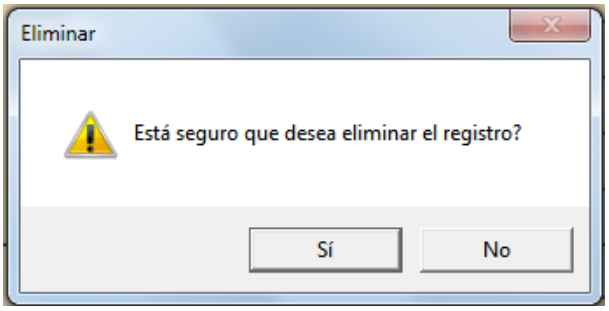
Al momento de buscar los datos nos aseguramos de guardar la dirección en la que esta ubicada dicho registro en una variable que llamamos posición, así que al momento de eliminar lo único que tenemos que hacer es ir a esa posición, seleccionarla y borrar el contenido.

```
Private Sub B_Eliminar_Clic()

    Dim Fila As String
    Dim Celdas As String
    Dim Nombre As String
    Dim x As Integer

Application.ScreenUpdating = False
    Nombre = T_Nombre
    Fila = Val(T_Posicion) + 5
    Celdas = "B" & Fila & ":" & "W" & Fila
    x = MsgBox("Está seguro que desea eliminar el registro?",
vbExclamation + vbYesNo, "Eliminar")

    If x = 6 Then
        Range(Celdas).Select
        Selection.Delete Shift:=xlUp
        MsgBox Prompt:="El empleado: " & Nombre & ", ha sido eliminado
del registro.", Buttons:=vbInformation, Title:="Eliminado"

'Borrar el contenido de los TextBox

Else
        Exit Sub

    End If
    Call B_BuscarNombres_Clic
    C_Nombre.Visible = False
    Application.ScreenUpdating = True
End Sub
```

La variable posición contiene la posición en la que se encuentra el registro a partir del primer registro, por eso para ubicar la fila correcta le sumamos cinco que corresponde al número de filas que no contienen datos.

Como en la mayoría de los programas aquí se solicita la confirmación para proceder a eliminar el registro.

Nota: El número 6 corresponde al botón sí de la función MsgBox.



GENERAR COLILLA

Con el botón Generar Colilla del formulario Consulta podemos generar la colilla de pagos de cada trabajador para imprimirla posteriormente con sólo hacer clic sobre el botón imprimir del mismo formulario.

El código de programación del botón Generar Colilla asigna los valores de cada uno de los *TextBox* del formulario Consulta a una hoja de cálculo de Excel donde se han definido los campos correspondientes a cada una de las variables que previamente han sido calculadas en el módulo "Crear Nomina". La colilla de pagos se muestra únicamente en la pantalla de vista previa de modo que los datos no pueden modificarse.

EMPRESA COMERCIALIZADORA DE NICARAGUA, S. A.
ECONICSA
COLILLA DE PAGO

Correspondiente a la 1ra quincena de Enero del 2010

PERIODO DE PAGO		NOMBRE	EMPLEADO No.	INSS #	CEDULA
DE	A				
01/01/2010	15/01/2010	Freddy Neftali Pineda Madrigal	123	123456	081-210488-0003F

INGRESOS

Salario Nominal	Antigüedad	Horas Extra	Comisiones	Otros Ingresos	TOTAL DEVENGADO
C$ 5,000.00	C$ 150.00	C$ 0.00	C$ 0.00	C$ 0.00	**C$ 5,150.00**

DEDUCCIONES

INSS	IR	Otras Deducciones	TOTAL DEDUCIDO
C$ 321.88	C$ 203.38	C$ 0.00	**C$ 525.26**

DETALLES

Detalles Otros	0	INSS Patronal:	824
		No. Horas Extra:	0

NETO A RECIBIR: C$ 4,624.74

RECIBI CONFORME: _____

Impreso el 20/01/2010 por: ECONICSA

IMPRIMIR COLILLA

Una vez generada la colilla de pagos y comprobada la información que contiene se puede proceder a imprimir dicha colilla con sólo hacer clic sobre el botón Imprimir colilla del antes mencionado formulario Consulta. Imprimir colilla no permite elegir una impresora sino que imprime en la impresora predeterminada por lo que si se tiene acceso a más de una impresora se debe establecer como impresora determinada aquella en la que deseamos imprimir las colillas de pago.

La parte esencial del código que imprime es el mismo utilizado para imprimir la base de datos de los empleados (*ActiveSheet.PrintOut*), de modo que el resto del código lo que hace es activar la hoja en la que se encuentran los datos y establecer el área de impresión.

2. Exportar

En ocasiones es necesario contar con la información de la planilla de pagos de tal modo que pueda ser manipulada para realizar cálculos y proyecciones por parte de la administración o gerencia de la empresa sin que ello afecte la base de datos que se maneja. Este programa incluye la opción de exportar a un nuevo libro de Excel la información con la que se prepara la planilla de pago de la empresa en el que se podrán hacer todas las modificaciones requeridas.

```
    Sheets(C_Mes & T_Hoja).Activate

    If Range("B6") = Empty Then
        MsgBox Prompt:="No hay registros en el Mes y Quincena que
seleccionó", Buttons:=vbExclamation, Title:="ATENCION!!!"
        Call Mostrar_Gestion
        Else
            Sheets(C_Mes & T_Hoja).Visible = True
            Sheets(C_Mes & T_Hoja).Select
            Sheets(C_Mes & T_Hoja).Copy

        End If
```

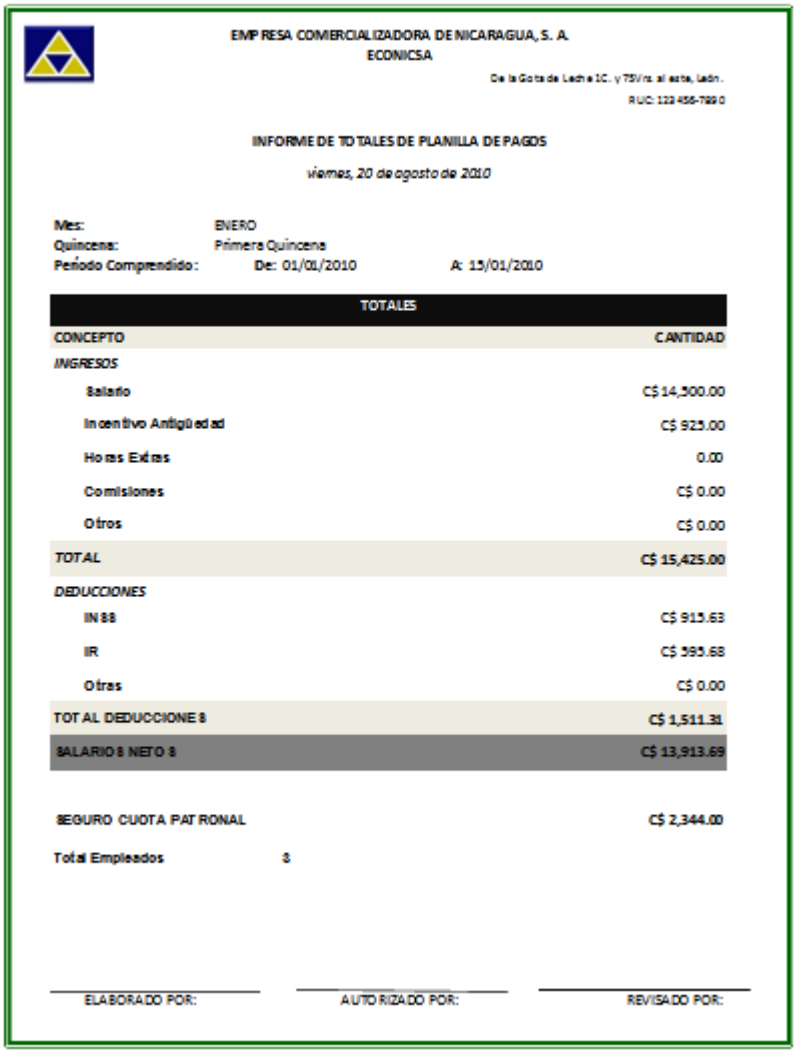
Con esta opción podemos exportar los datos correspondientes a una quincena a la vez, es decir, la que se elija en el formulario Gestión.

La primera línea de código mostrada activa la hoja que contiene los datos que se desean exportar, luego se verifica que existen datos en dicha hoja y si no hay datos se muestra un mensaje indicando que no hay datos para el periodo seleccionado y el programa se sale del procedimiento.

3. Generar informe

En esta parte del programa se ha diseñado un informe que muestra un consolidado de los distintos conceptos que integran una planilla de pagos con la estructura que se muestra en la imagen a continuación.

Este es un informe requerido tanto por la dirección de la empresa para efectos contables como por órganos que de alguna manera están involucrados en las relaciones obrero-patronales (Seguro social, Dirección General de Ingresos, Ministerio del trabajo).



EMPRESA COMERCIALIZADORA DE NICARAGUA, S. A.
ECONICSA
De la Gota de Leche 1C. y 75V rs. al este, León.
RUC: 123 456-7890

INFORME DE TOTALES DE PLANILLA DE PAGOS

viernes, 20 de agosto de 2010

Mes: ENERO
Quincena: Primera Quincena
Período Comprendido: De: 01/01/2010 A: 15/01/2010

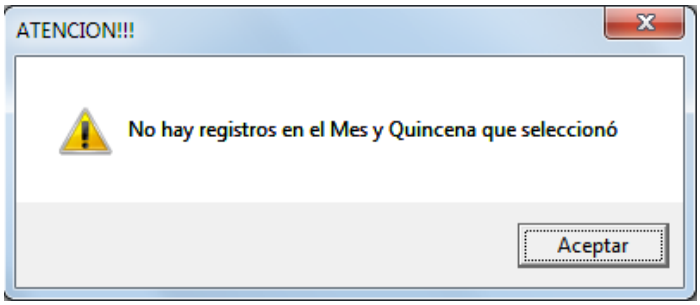
TOTALES	
CONCEPTO	CANTIDAD
INGRESOS	
Salario	C$ 14,500.00
Incentivo Antigüedad	C$ 925.00
Horas Extras	0.00
Comisiones	C$ 0.00
Otros	C$ 0.00
TOTAL	C$ 15,425.00
DEDUCCIONES	
INSS	C$ 913.63
IR	C$ 595.68
Otras	C$ 0.00
TOTAL DEDUCCIONES	C$ 1,511.31
SALARIOS NETOS	C$ 13,913.69
SEGURO CUOTA PATRONAL	C$ 2,344.00
Total Empleados	3

ELABORADO POR: AUTORIZADO POR: REVISADO POR:

```
Sheets(C_Mes & T_Hoja).Activate

If Range("B6") = Empty Then

  MsgBox Prompt:="No hay registros en el Mes y Quincena que  seleccionó",
Buttons:=vbExclamation, Title:="ATENCION!!!"

Else

  Hoja3.Range("D13").Value = Range("W4").Value
  Hoja3.Range("F13").Value = Range("X4").Value
  Hoja3.Range("F18").Value =
  WorksheetFunction.Text(WorksheetFunction.Sum(Range("D6:D400")),
  "#,###0.00")
  Hoja3.Range("F19").Value =
  WorksheetFunction.Text(WorksheetFunction.Sum(Range("E6:E400")),
  "#,###0.00")
```

```
    Hoja3.Range("F20").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("F6:F400")),
    "#,###0.00")

    Hoja3.Range("F21").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("G6:G400")),
    "#,###0.00")

    Hoja3.Range("F22").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("H6:H400")),
    "#,###0.00")

    Hoja3.Range("F23").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("I6:I400")),
    "#,###0.00")

    Hoja3.Range("F25").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("J6:J400")),
    "#,###0.00")

    Hoja3.Range("F26").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("K6:K400")),
    "#,###0.00")

    Hoja3.Range("F27").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("L6:L400")),
    "#,###0.00")

    Hoja3.Range("F28").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("M6:M400")),
    "#,###0.00")

    Hoja3.Range("F29").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("N6:N400")),
    "#,###0.00")

    Hoja3.Range("F30").Value =
    WorksheetFunction.Text(WorksheetFunction.Sum(Range("O6:O400")),
    "#,###0.00")

    Hoja3.Range("D32").Value = WorksheetFunction.CountA(Range("B6:B400"))

    Sheets("INFORME").Activate
      ActiveWindow.SelectedSheets.PrintPreview

End If
```

La primera parte del bloque de código activa la hoja que contiene los datos, es decir, la quincena de la cual deseamos crear el informe y verificar que existen datos para dicho periodo, de no ser así se mostraría un mensaje indicándonos que no hay datos para el periodo seleccionado.



La siguiente parte se encarga de sumar los rangos de celdas donde se encuentran los datos correspondientes a las asignaciones y deducciones de todos los empleados a los que se ha remunerado en la quincena y escribir el resultado en la hoja en la que se prepara el informe. La última parte del código activa la vista previa del informe en un formato para imprimirse.

4. Generar planilla

La cuarta opción que brinda el formulario de Gestión es obtener el informe final con el detalle de las asignaciones y deducciones de todos y cada uno de los empleados que laboran para la empresa.

Aquí se brinda la oportunidad al usuario de obtener en una hoja, configurada para imprimirse, la planilla de pagos o nómina que es el propósito final de nuestro trabajo y aunque la parte esencial está en realizar los cálculos, es aquí donde se ve el resultado.

Recordar una vez más que éste, al igual que los informes descritos anteriormente, se deben generar una vez que se hayan hecho los cálculos para todos los trabajadores.

```
    Sheets(C_Mes & T_Hoja).Activate

    Range("B6").Activate

    If Range("B6") = Empty Then
        MsgBox Prompt:="No hay registros en el Mes que seleccionó",
Buttons:=vbInformation, Title:="ATENCION!!!"
        Load Gestion
        Gestion.Show

    Else

    Do While Not IsEmpty(ActiveCell)
        ActiveCell.Offset(1, 0).Activate
        Filas = ActiveCell.Offset(0, -1).Value + 4
    Loop

    Range("B6").Activate
    Rango_Datos = "$B$6:$W$" & Filas
    Range(Rango_Datos).Copy
    Hoja9.Range("$Q$8:$AL$" & Filas + 2).PasteSpecial (xlPasteValues)
    Rango_Datos = "$A$1:$N$" & Filas + 2
```

```
Hoja9.Range("B7").Value = "Número de Empleados: " &
WorksheetFunction.CountA(Range("B6:B400"))

Hoja9.Range("C7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("D6:D400")),
"#,###0.00")

Hoja9.Range("D7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("E6:E400")),
"#,###0.00")

Hoja9.Range("E7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("F6:F400")),
"#,###0.00")

Hoja9.Range("F7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("G6:G400")),
"#,###0.00")

Hoja9.Range("G7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("H6:H400")),
"#,###0.00")

Hoja9.Range("H7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("I6:I400")),
"#,###0.00")

Hoja9.Range("I7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("J6:J400")),
"#,###0.00")
Hoja9.Range("J7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("K6:K400")),
"#,###0.00")

Hoja9.Range("K7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("L6:L400")),
"#,###0.00")

Hoja9.Range("L7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("M6:M400")),
"#,###0.00")

Hoja9.Range("M7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("N6:N400")),
"#,###0.00")

Hoja9.Range("N7").Value =
WorksheetFunction.Text(WorksheetFunction.Sum(Range("O6:O400")),
"#,###0.00")

    Hoja9.Activate
    ActiveSheet.PageSetup.PrintArea = Rango_Datos

    ActiveWindow.SelectedSheets.PrintPreview
    Hoja9.Range("$Q$8:$AL$" & Filas + 2).Clear

End If
```

El código con el cual generamos la planilla de pagos lo que hace es activar la hoja en la que se encuentran los datos de la quincena que nos interesa, verifica que hay datos (si no hay datos se muestra un mensaje y el procedimiento nos regresa al inicio), los copia y luego suma para obtener los totales de cada una de las variables involucradas (salarios comisiones, cuota del seguro, etc.) y a continuación se muestra la vista previa de la planilla de pagos con la estructura que se muestra en la imagen.

15/09/2010

EMPRESA COMERCIALIZADORA DE NICARAGUA, S. A.
ECONICSA
RUC: 123456
ENERO, Primera Quincena

RUC: 123456-7890
De la Gota de Leche 1C. y 75Vrs. al este, León.

Frente a Iglesia la Merced

CODIGO	Nombres y Apellidos del Empleado	Sueldo Mensual	Incentivo Antigüedad	Horas Extra	Comisiones	Otros Ingresos	TOTAL DEVENGADO	INSS	IR	Otras Deducciones	TOTAL DEDUCCIONES	Neto a Pagar	Seguro Cuota Patronal
Totales:	*Número de Empleados: 4*	*14.505,00*	*925,00*	*0,00*	*0,00*	*0,00*	*15.430,00*	*915,94*	*595,68*	*0,00*	*1.511,62*	*13.918,38*	*2.344,80*
456	Nelson Enrique Jarquin Canales	6.000,00	600,00	0,00	0,00	0,00	6.600,00	375,00	367,92	0,00	742,92	5.857,08	960,00
12345	Jael Georgina Meza Carrero	8.300,00	175,00	0,00	0,00	0,00	8.475,00	218,75	24,38	0,00	243,13	8.431,87	760,00
123	Freddy Neftalí Pineda Madrigal	5.000,00	150,00	0,00	0,00	0,00	5.150,00	321,88	203,38	0,00	525,26	4.624,74	824,00
321	Elena Toruño	5,00	0,00	0,00	0,00	0,00	5,00	0,31	0,00	0,00	0,31	4,69	0,80

VII. CONCLUSIONES

El propósito con el que hemos creado este programa es dar repuesta a la problemática que enfrenta toda empresa en el manejo de la información concerniente a los datos laborales de sus emplados cuando no posee un sistema informatizado que permita manejar de forma organizada los datos utilizados en la elaboración de la planilla de pagos.

Hemos logrado documentar los aspectos generales utilizados en la elaboración y que forman parte de la planilla de pagos de cualquier empresa establecida en el país, de acuerdo a las reformas que recientemente se efectuaron en materia fiscal. Aunque recalcamos que hay aspectos muy particulares, propios de cada empresa, en relación a los ingresos y deducciones salariales de sus empleados, por ejemplo el incentivo por antigüedad el cual se otorga según políticas establecidas por cada empresa pues no existe una normativa al respecto.

En el desarrollo de esta investigación logramos dar una explicación paso a paso de cómo se pueden utilizar algunas funciones y herramientas de programación para crear una aplicación práctica y funcional para resolver problemas de todo tipo y tan complejas como el manejo de una base de datos. Si bien es cierto que existe software de última tecnología que se puede utilizar para gestionar la planilla de pagos de una empresa el costo de adquisición es alto y las empresas pequeñas o medianas no podrían costear así que el recurrir a un sistema como el que hemos presentado puede simplificar significativamente las tareas diarias en el entorno laboral.

Finalmente, queremos mencionar que este trabajo es producto de la dedicación a la investigación y el autoaprendizaje y como tal esperamos despertar el interés de estudiantes, profesores y profesionales en general por la investigación, ya que por experiencia propia sabemos que la reconmpensa es gratificante.

VIII. RECOMENDACIONES

Hemos hecho un gran esfuerzo por aplicar los conocimientos adquiridos para la creación de este sencillo pero muy práctico sistema, y aunque tratamos de incluirle las funciones básicas para facilitar el trabajo específico del control de nómina de una empresa, estamos consientes de que se le pueden hacer algunas mejoras, o hasta correcciones. Puesto que nuestro trabajo puede servir de referencia para proyectos posteriores semejantes, plantearemos algunas mejoras que recomendamos se tengan presente para futuros proyectos.

Incluir al sistema una modalidad para llevar el control de vacaciones de los empleados, liquidaciones, deducción por llegadas tarde o inasistencias y pago de aguinaldo, en vista de que hay empresas que podrían requerir de estas funciones.

Crear una conexión a una Base de Datos de Microsoft Access, por ejemplo, o SQL, que igualmente se pueda trabajar desde Excel, esto da mayor seguridad a la base de datos ante la posible pérdida de la información.

Agregar un módulo para generar algunos informes contables y un reporte al final del año fiscal.

IX. BIBLIOGRAFIA

Libros:

- Báez Cortés, Theódulo; Báez Cortés, Julio Francisco. *Todo sobre impuestos en Nicaragua*. 6ta ed. Managua, Nicaragua: Instituto Nicaragüense de Investigaciones y Estudios Tributarios. 2004. 737 p.

- Warren, Carls; James, M. *Contabilidad Financiera*. 9na ed. [s.l.] Thomson Internacional, 2005. 830 p.

Leyes y Reglamentos:

- Nicaragua. *Código del trabajo*. La Gaceta, 30 de Octubre de 1996, No. 205

- Nicaragua. *Ley de reformas y adiciones a la ley No. 453, Ley de equidad fiscal y a la ley No. 528, Ley de reformas y adiciones a la Ley de equidad fiscal.* La Gaceta, 21 de Diciembre de 2009, No. 241

- Nicaragua. *Ley de creación de la planilla nacional de pago.* La Gaceta, 03 de Enero de 1983, No. 01

- Nicaragua. *Reforma al decreto No. 975, Reglamento general de la ley de seguridad social*. La Gaceta, 07 de Enero de 2010, No. 04

- *Ley de equidad fiscal*. Disponible en Web: http://www.dgi.gob.ni/documentos/Ley%20de%20Equidad%20Fiscal%20Ley%20No.%20453.PDF

- *Ley del impuesto sobre la renta.* Disponible en Web: http://www.dgi.gob.ni/documentos/Ley%20del%20Impuesto%20Sobre%20la%20Renta%20Decreto%20No.%20662.PDF

- *Reglamento del impuesto sobre la renta.* Disponible en Web: http://www.oas.org/juridico/spanish/mesicic3_nic_imp_renta.pdf

Artículos en internet:

- Matilde Carolina Medina. *Derechos laborales-nómina de pago.* http://www.monografias.com/trabajos61/derechos-laborales/derechos-laborales.shtml

- *Manual de visual basic para excel.* http://www.manualespdf.es/manual-programacion-visual-basic-excel/

- *La nómina.* Rosanna Silva Fernandez. http://www.monografias.com/trabajos42/nomina/nomina.shtml

Manual de procedimientos. Ministerio de educación. http://www.mined.gob.ni/PDF07/MANUAL%20DE%20FUNCIONES/Procedimientos%20Division%20RR%20HH-b.pdf

Otros:

Báez Cortés, Theódulo; Chavarría, Dennis. *Seminario-Taller sobre la reforma a la Ley 453, Ley de Equidad Fiscal.* BAEZ CORTEZ Y CIA. LTDA. Managua, 25 de Enero de 2010.

Archivo de Ayuda de Microsoft Excel 2007 y de la *Ayuda de Visual Basic 6.3.* Versión de Octubre del año 2006.

ANEXOS