Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO AUTOMATIZADO Y ESTRUCTURADO DE LA PRÁCTICA DE DESTILACIÓN DE UNA SOLUCIÓN BINARIA -LABORATORIO DE OPERACIONES UNITARIAS-

WALTER GIOVANNI ALVAREZ MARROQUIN

ASESORADO POR: ING. GUSTAVO A. VILLEDA VÁSQUEZ

GUATEMALA, AGOSTO DE 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO AUTOMATIZADO Y ESTRUCTURADO DE LA PRÁCTICA DE DESTILACIÓN DE UNA SOLUCIÓN BINARIA -LABORATORIO DE OPERACIONES UNITARIAS-

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

WALTER GIOVANNI ALVAREZ MARROQUIN
ASESORADO POR: ING. GUSTAVO A. VILLEDA VÁSQUEZ

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO ELÉCTRICO

GUATEMALA, AGOSTO DE 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	
VOCAL II	Lic. Amahán Sánchez Alvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADOR	Ing. Armando Gálvez Castillo
EXAMINADOR	Inga. Ingrid Salome Rodriguez de Loukota
EXAMINADOR	Ing. Marvin Marino Hernández Fernandez
SECRETARIA	Ing. Pedro Antonio Aguilar Polanco

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO AUTOMATIZADO Y ESTRUCTURADO DE LA PRÁCTICA DE DESTILACIÓN DE UNA SOLUCIÓN BINARIA
-LABORATORIO DE OPERACIONES UNITARIAS-

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha febrero de 2005.

Walter Giovanni Alvarez Marroquín

Guatemala, 12 de julio de 2005

Señor Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Señor Coordinador:

Por medio de la presente, me permito informarle que he revisado completamente el trabajo de tesis titulado: "DISEÑO AUTOMATIZADO Y ESTRUCTURADO DE LA PRÁCTICA DE DESTILACIÓN DE UNA SOLUCIÓN BINARIA (LABORATORIO DE OPERACIONES UNITARIAS)"; desarrollado por el señor Walter Giovanni Alvarez Marroquín y puedo concluir que dicho trabajo cumple con los objetivos propuestos en el anteproyecto.

Por lo tanto, el autor del trabajo de graduación y yo, como su asesor, nos hacemos responsables por el contenido y conclusiones de la misma.

Atentamente

Ing. Gustavo Adolfo Villeda Vásquez
Asesor Nombrado
Colegiado 3654

AGRADECIMIENTOS

A:

DIOS	Por permitirme iniciar una vida de servicio profesional.
mis padres	Por su apoyo diario y comprensión en las etapas de mi vida que me han acompañado, y este paso es algo que lo hemos logrado juntos por la divina gracia de Dios.
mis catedráticos	Les agradezco por haber contribuido en mi formación profesional y en especial al Ing. Gustavo Villeda por sus consejos y su amistad brindada, y el apoyo para poder desarrollar este trabajo de graduación.
la universidad	Por darme la dicha de pertenecer a un grupo selecto de profesionales.
mis compañeros	Alejandro, Nico, Werner, Luis, Walter y en especial a Emmet Echeverría por su apoyo en este trabajo de graduación.

DEDICATORIAS:

A:

DIOS	Por concederme la gracia de terminar mis estudios.
mis padres	Ricardo Raúl Alvarez Soto (†) y Dominga del Pilar Marroquín Castillo por formarme con su ejemplo en un marco de sencillez, honradez, trabajo y dedicación.
mis hermanos	Ricardo, Karla y Miriam.
mi novia	Luz Maria por su amor y apoyo incondicional.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

GLOSARIO

ALU	Unidad encargada de realizar operaciones del tipo aritmético-lógicas.
Algoritmo	Secuencia ordenada de pasos exenta de ambigüedades, que lleva a la solución de un problema dado.
Alfanumérico	Referente a un conjunto de caracteres que contiene letras, dígitos, símbolos de puntuación y especiales.
Aplicación	Problema o tarea en el que puede utilizarse o aplicarse la computadora
ASCII	American Standard Code for Information Interchange: Código estándar norteamericano para intercambio de información, sistema de codificación de 7 bits.
Bit	Dígito binario (0 ó 1)
Buffer	Área intermedia de almacenamiento utilizada para compensar las referencias en las razones de flujo de datos o en los tiempos de ocurrencia de eventos, al transmitir datos de un dispositivo a otro
Byte	Grupo de bits adyacentes configurado para representar datos alfanuméricos

Carácter	Una unidad de dato alfanumérico.
Ciclo	Secuencia de instrucciones de programa que se ejecutan en forma repetida hasta que se cumpla una condición especifica.
Código	1) Reglas utilizadas para traducir una configuración de bits a caracteres alfanuméricos. 2) Proceso de recopilar instrucciones de computadora en forma de programa de computadora. 3) el propia programa de computadora.
Computadora	Lo mismo que procesador.
Computadora personal	PC: personal computer, microcomputadora de costo relativamente bajo.
Control	Referente a los procedimientos utilizados para asegurar la integridad de la entrada a un sistema de información y la salida proveniente de éste.
Control de proceso	Uso de la computadora para controlar un proceso en marcha en un ciclo continúo de retroalimentación.
Depurar	Eliminar los “errores ocultos”
Diseño estructurado	Referente a la Modularización de un programa o sistema en unidades lógicas funcionales utilizando el enfoque de diseño descendente.

Diseño modular	Descomposición en segmentos de un programa, para después analizarlo y diseñarlo, o descomposición de un sistema en módulos inteligibles.
Dispositivo de E/S	Dispositivo de hardware capaz de manejar entrada y/o salida proveniente del procesador o hacia este.
En línea	Referente a datos y/o dispositivos de hardware que estén accesibles a un sistema de computadora o bajo el control de éste
Entrada	Datos que deben procesarse en un sistema
Hardware	Los dispositivos físicos que constituyen un sistema de computo
Instrucción	Enunciado de programa que especifica que debe realizarse una operación específica de computadora.
Lenguaje de programación de alto nivel	Lenguaje de programación no dependiente de la computadora, el cual utiliza una sintaxis como la del idioma inglés, en la que cada enunciado corresponde a muchas instrucciones del lenguaje ensamblador. Los lenguajes de alto nivel [*high-level languages*] liberan a los programadores de tener que lidiar con la arquitectura subyacente de la máquina, y les permite concentrarse en la lógica del problema que tratan de resolver. En la práctica, todos los lenguajes de programación con más recursos operacionales que un lenguaje ensamblador (o de ensamble) se consideran lenguajes de alto nivel.

Lenguaje ensamblador	Lenguaje simbólico de bajo nivel con un conjunto de instrucciones que esencialmente guarda relación uno a uno con el lenguaje de máquina.
Mantenimiento	Proceso continúo a través del cual los sistemas se actualizan y mejoran para cubrir necesidades cambiantes.
Procesador	El componente lógico de un sistema de computadora que interpreta y ejecuta las instrucciones del programa.
Programa	Instrucciones de computadora estructuradas y ordenadas deforma tal que al ejecutarse hacen que una computadora realice alguna función especifica.
Programación estructurada	Modularización de un programa en una de las tres construcciones básicas de programa (secuencia, decisión y ciclo) y la aplicación de principios de diseño estructurado.
Recorrido estructurado	Procedimiento de evaluación de emulación para programas y sistemas en desarrollo. Se utiliza para minimizar la posibilidad de que algo se haya olvidado o se esté haciendo mal.
Salida	Datos transferidos a partir del almacenamiento primario a un dispositivo de E/S.
Seudocodigo	Código no ejecutable de programa que se utiliza como auxiliar para desarrollar y documentar programas estructurados.
Sistema	La integración de personas, procedimientos, software y/o hardware para lograr una función especifica.

Sistema operativo	OS: Operating system, el software que controla la ejecución de todas las aplicaciones y programas de software de sistemas.
Software	Programas utilizados para dirigir las funciones de un sistema de cómputo.

RESUMEN

Este trabajo de graduación presenta una serie de consideraciones de los aspectos más relevantes, empleados en el diseño de hardware y software, necesarios para la implementación de proyectos industriales automatizados, mostrando los principios y técnicas utilizadas en el diseño de la práctica de destilación de una solución binaria y las ventajas que ofrecen los métodos de diseño estructurados.

El primer capítulo describe métodos de diseño estructurados que facilitarán el inicio y desarrollo de proyectos industriales automatizados.

El segundo capítulo describe principios y técnicas de programación que facilitarán la codificación, la prueba y el refinamiento del software y hardware necesarios para la correcta automatización de la práctica de destilación binaria.

El tercer capítulo desarrolla, completamente, la automatización de la práctica de destilación binaria, formando paso a paso la interfase visual, programa que se ejecutará en una computadora tipo PC y el programa lógico interno en un autómata programable S7-200; la comunicación entre ambos logrará realizar la práctica más convenientemente.

El cuarto capítulo describe la forma de integrar la práctica de destilación de una solución binaria, mostrando las diferentes etapas de ejecución del diseño.

OBJETIVOS

- **OBJETIVO GENERAL**

  Diseñar la práctica de destilación de una solución binaria en el laboratorio de operaciones unitarias de la facultad de ingeniería de la Universidad de San Carlos de Guatemala, en forma automatizada y estructurada.

- **OBJETIVOS ESPECIFICOS**

  1. Dar a conocer métodos estructurados que nos permitan facilitar la tarea de diseñar hardware y software, para su utilización en un sistema.

  2. Sintetizar principios de una buena programación, aplicados especialmente a los proyectos grandes, reforzando el diseño, la codificación, la verificación y la documentación de programas.

  3. Diseñar una solución automatizada y estructurada usando lenguajes de programación de alto nivel, Microsoft: Visual Basic, Siemens: STEP 7 MicroWin, estructurados que refuercen los conceptos de programación y diseño Modular.

INTRODUCCIÓN

Un sistema automatizado y estructurado es una combinación de técnicas de diseño e implementación, totalmente, eficiente y moderna. La creciente demanda y evolución que, en forma acelerada, está sufriendo la industria en el campo de la producción, ha permitido que la preparación e investigación de las prácticas realizadas por los futuros profesionales sean tan exactas y accesibles como sea posible para su futura aplicación en la industria.

El estudio de un sistema automatizado y estructurado, permite agilizar el acceso a un proceso de producción para personas particulares y empresas. La intervención humana se hace mínima porque ya no es necesario estar presente durante todo el proceso.

Un proceso manual carece de flexibilidad, rapidez y exactitud. Una automatización eficiente y estructurada permitirá ejecutar los procesos de la práctica de destilación de una solución binaria, más cómodamente, tomándose la libertad de seleccionar la potencia en el calderín y la toma de los datos de temperatura en cada plato de la torre desde un programa de tipo visual, Visual Basic, que se ejecutará desde una computadora con Windows como sistema operativo, lo cual permitirá que el intercambio de información entre el autómata programable, S7-200, y la computadora haga posible el trabajo que antiguamente se hacía, manualmente.

A lo largo de este documento, se dan a conocer métodos estructurados y principios de una buena programación que permitirán facilitar la tarea de diseñar hardware y software aplicados, especialmente, a los proyectos grandes, mostrando al final una solución automatizada y estructurada de la práctica de destilación de una solución binaria.

1. DISEÑO ESTRUCTURADO

1.1. Algoritmo

En la actualidad, gran cantidad de programas son ejecutados. Debido a que los mismos requieren de un equipo de personas; la comunicación y la compactibilidad que existen entre ellas es cada día más importante.

Un algoritmo puede ser utilizado para realizar el diseño de un programa o bien para realizar una secuencia de actividades a ejecutar en forma ordenada.

Algoritmo puede definirse como una secuencia ordenada de pasos exenta de ambigüedades, que lleva a la solución de un problema dado. Las direcciones dadas a una calle constituyen un algoritmo para encontrar la calle; una receta es una forma muy común de algoritmo; los planos sirven el mismo propósito en una obra de construcción.

Se exigirá que los algoritmos tengan varias propiedades importantes: primero, los pasos de un algoritmo deben ser simples y exentos de ambigüedad y seguir un orden cuidadosamente prescrito. Además, se insistirá en que los algoritmos sean efectivos, o sea, deben resolver siempre el problema en un número finito de pasos.

Como ejemplo, se diseñara un algoritmo para cambiar una bombilla quemada.

El algoritmo para reemplazar la bombilla quemada consta de siete pasos.

1. Sitúese la escalera debajo de la bombilla quemada.
2. Elíjase una nueva bombilla de la misma potencia que la anterior.

3.Súbase por la escalera hasta que pueda alcanzar la bombilla

4.Gírese la bombilla en sentido antirreloj hasta que esté suelta

5.Ubíquese la nueva en el mismo lugar

6.Enrósquese en el sentido de las manecillas del reloj hasta que quede apretada

7.Bájese de la escalera.

Pero aún no se han elaborado las instrucciones con bastante precisión. Varios de los pasos de este algoritmo implican operaciones más detalladas que deben indicarse específicamente.

1. Sitúese la escalera debajo de la bombilla quemada.
2. Elíjase una posible reemplazante.

   Si la potencia no es igual a la antigua, repítase el siguiente procedimiento hasta que sea igual.

   Descártese la bombilla elegida.

   Elíjase una nueva.
3. Repítase el procedimiento hasta que se alcance la bombilla quemada.

   Súbase un escalón tras otro de la escalera.
4. Repítase el siguiente procedimiento hasta que la bombilla esté suelta.

   Gírese la bombilla en sentido antirreloj.
5. Ubíquese la nueva en el mismo lugar
6. Repítase el siguiente procedimiento hasta que la bombilla esté apretada.

   Enrósquela en el sentido de las manecillas del reloj.
7. Bájese de la escalera.

El algoritmo del ejemplo para cambiar la bombilla quemada, sirve para ejemplificar la forma en que deben expresarse los algoritmos. Las operaciones indicadas son simples y están exentas de ambigüedad, las decisiones que se piden son simples y se especifica claramente el orden en que se deben efectuar los pasos.

El método por medio del cual este algoritmo fue desarrollado es importante; se comenzó con un enunciado muy general de la solución del problema y se continuó hasta encontrar el algoritmo final incrementando sistemáticamente los detalles.

¿Cómo se sabe cuándo hay un nivel adecuado en los detalles de un algoritmo? Está es una función que debe realizar quien utilizará el algoritmo. Si se está dando una receta al cocinero jefe de un gran hotel, debe suponerse que está familiarizado con muchos aspectos del arte culinario, los cuales no conoce la persona que está poniendo a hervir su primera olla de agua.

La receta debe expresarse en tales términos que sea entendida por la persona concreta que va a preparar la comida, lo cual también es cierto para los algoritmos que se elaboran para ser procesados mediante computadora. Estas máquinas tienen un conjunto muy limitado de instrucciones que pueden realizar y el algoritmo debe, necesariamente, expresarse en términos de ellas.

La expresión de un algoritmo es muy importante, porque ayuda no sólo a la traducción que se haga al lenguaje de máquina, sino también al desarrollo del algoritmo mismo.

1.2. Descripción de algoritmos

El desarrollo de un algoritmo apropiado es la primera etapa de la preparación de un programa de computación para una aplicación específica. En la sección 1.1 se vio que un algoritmo es una secuencia de pasos que deben seguirse en un orden cuidadosamente preestablecido. Se hizo hincapié en la importancia de la precisión. Los pasos del algoritmo deben expresarse claramente y sin ambigüedades.

La descripción de un algoritmo en esta forma clara y fácil de seguir sirve de gran ayuda en su desarrollo, su consiguiente transformación en un programa para ser ejecutado y la documentación del mismo una vez que ha sido concluido.

1.2.1. Descripción narrada

Un método sencillo de expresar un algoritmo es relatarlo verbalmente. El lenguaje natural es farragoso e impreciso y con frecuencia es un vehículo muy poco confiable para transmitir información. La compresibilidad se sacrifica en aras de la precisión.

Debido a la imprecisión del lenguaje natural, el peligro de malinterpretar o de perder información es muy grande. Las interpretaciones sutiles y los sentidos del significado pueden estimular la imaginación del lector de novelas, pero en los escritos técnicos ensombrecen los detalles y llevan rápidamente a errores.

Por estas razones, el lenguaje natural no es el vehículo apropiado, cuando se le usa sólo para expresar los algoritmos. A pesar de estos inconvenientes, la prosa es útil para presentar las notas aclaratorias o para destacar los casos especiales. Por esta razón, no se descartará totalmente la narración descriptiva en al búsqueda de mejores métodos de expresión.

1.2.2. El diagrama de flujo

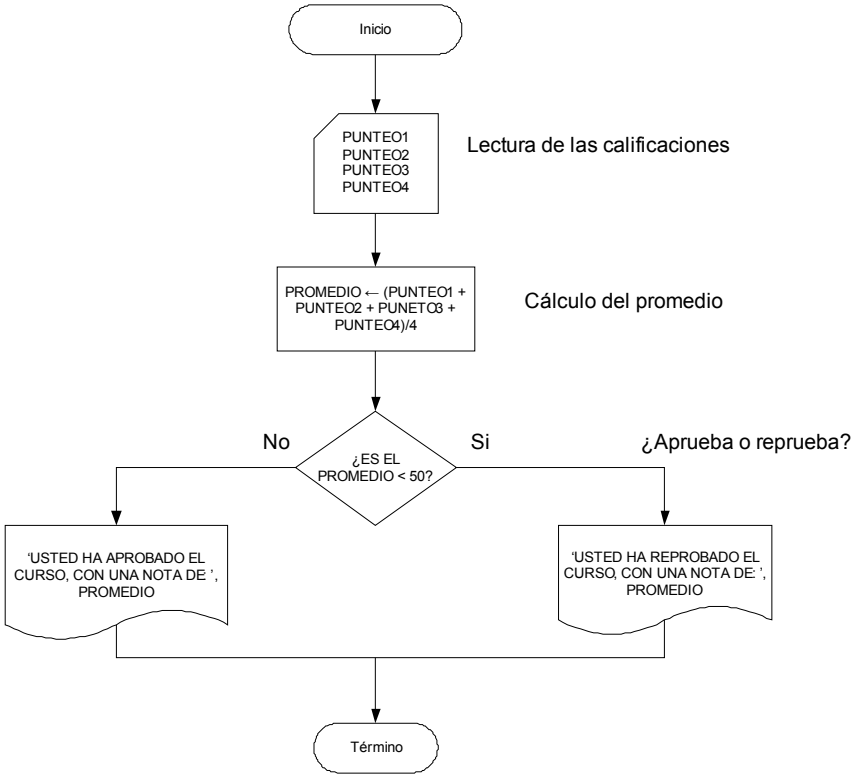
Con frecuencia es más sencillo expresar ideas gráficamente que en forma verbal. Un intento de proporcionar una visión gráfica de la descripción de los algoritmos, que data desde los tiempos de *Von Neumann*, lo constituye el uso de los diagramas de flujo. Un diagrama de flujo muestra la lógica del algoritmo, acentuando los pasos individuales y sus interconexiones. Con los años ha surgido un simbolismo relativamente estandarizado (véase la figura 1).

FIGURA 1: Símbolos comunes de los diagramas de flujo

Los cálculos se representan con un rectángulo, las decisiones con un rombo, y las operaciones de entrada y salida se simbolizan con figuras esquemáticas de los medios utilizados.

Las figuras se conectan directamente mediante líneas que indican el orden en el cual deben realizarse las operaciones. Un diagrama de flujo simple, con anotaciones, se muestra en la figura 2.

FIGURA 2: Diagrama de flujo simple



1.2.3. Un lenguaje algorítmico

Un Pseudocódigo es una forma de representar un programa. Consiste en una serie de instrucciones, que no necesariamente deben de pertenecer a un lenguaje de programación establecido, pero que no son ambiguas, ya que éstas deben de ser interpretadas más adelante por el diseñador o alguna otra persona.

Para realizar un Pseudocódigo sólo se debe saber que existen instrucciones lógicas y procedimientos o subrutinas. Un factor importante en su realización, son las sangrías y el lenguaje utilizado, ya que de ello depende su fácil entendimiento; si se utiliza alguna instrucción especial, es necesario definirla, con el fin de que no se pierda la lógica del procedimiento. Del diagrama de flujo de la figura 2 el algoritmo con pseudocódigos sería el siguiente:

```
Procedimiento NOTAS
Inicio
        Read (PUNTEO1, PUNTEO2, PUNTEO3, PUNTEO4)
        PROMEDIO ← (PUNTEO1, PUNTEO2, PUNTEO3, PUNTEO4)/4
        If PROMEDIO < 50
        Then
          Write (‘Usted ha reprobado el curso, con una nota de: ‘, PROMEDIO)
        Else
                Write (‘Usted ha aprobado el curso, con una nota de: ‘, PROMEDIO)
Fin
```

1.2.4. Resumen de la metodología de solución

Al enfrenar un problema para ser resuelto con la computadora, se sugiere que se sigan los siguientes pasos:

1. Asegúrese de entender completamente las especificaciones del problema. Una práctica común es indicar la salida que se espera de algunas entradas de ejemplo.

2. Formúlese grosso modo un algoritmo general para resolver el problema, poniendo muy poca atención en la especificación de los detalles. Esto constituye el primer intento de expresar la estrategia de la solución. Asegúrese de que esta estrategia es correcta probando los diferentes pasos con algunos ejemplos de entrada. Si no lo es, vuélvase a revisar.
3. Identifíquese y lístese cualquier variable que parezca ser necesaria. Inclúyase junto con el nombre de la variable su tipo y una indicación de su propósito.
4. Regrésese a los pasos individuales del algoritmo y procédase a llenar los detalles. Cada vez que un paso se divide en un número mayor de dos o más pasos de detalle, asegúrese de que los nuevos pasos más detallados realizan la función que se expresó en el paso original.
5. Cuando crea haber producido un algoritmo lo bastante detallado, pruébese con cuidado los pasos con algún conjunto de datos de entrada hasta estar seguro de que la solución elaborada satisface los requisitos establecidos.
6. Sólo cuando se ha efectuado lo anterior será el momento de vaciar el algoritmo en algún lenguaje de programación en particular.

1.3. Estructuras de decisión

Se ha visto la forma de desarrollar algoritmos para resolver cierta clase de problemas. Los algoritmos seguían más o menos una pauta determinada: la entrada de algunos datos, una serie de cálculos y la salida de los resultados. Sin embargo, la mayor parte de la potencia de la computadora proviene de su capacidad de tomar decisiones y determinar un sentido de acción en el momento de la ejecución, sobre la base de los valores de algunos elementos de los datos que se leen o de los resultados de algunos cálculos.

El formato de la construcción *if-then-else* es el siguiente:

If "condición"

Then

"alternativa verdadera"

Else

"alternativa falsa"

En la figura 3 y 4 muestran el diagrama de flujo general de la construcción de selección.

FIGURA 3: Diagrama de flujo, *if then else*

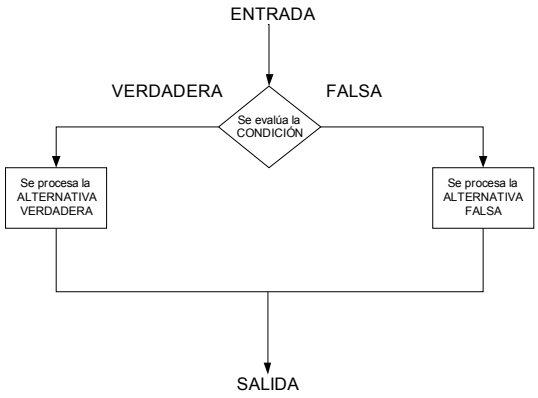
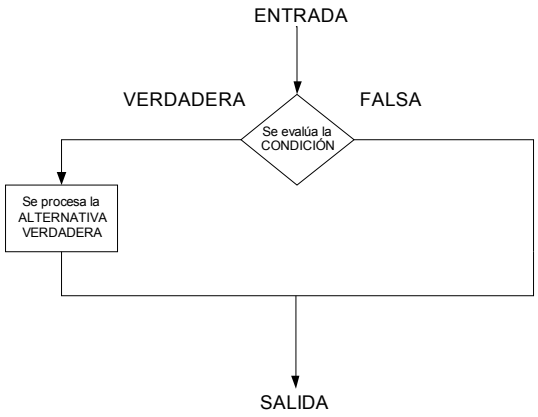


FIGURA 4: Diagrama de flujo sin alternativa falsa.

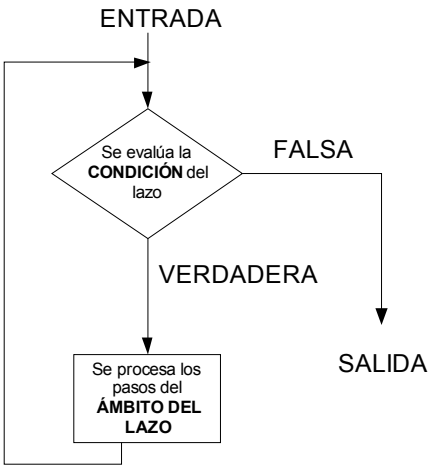


Las estructuras de repetición permiten que el programador especifique que la acción debe repetirse mientras cierta condición permanece verdadera.

Repeat
Steps
While "condición"

La estructura de control *repeat* se considerará como una sola unidad que tiene un simple punto de entrada y un simple punto de salida, tal como se muestra en la figura 5. La entrada y la salida se producen por el propio enunciado de repetición. Se trata del primer enunciado en ser ejecutado y también el último. Esto quiere decir que la condición que se expresa en le enunciado de repetición se ejecuta antes que cualquiera de los pasos del intervalo del lazo, y que la salida tiene lugar solamente cuando la condición a evaluar resulte falsa.

FIGURA 5: Diagrama general de flujo de la construcción lazo

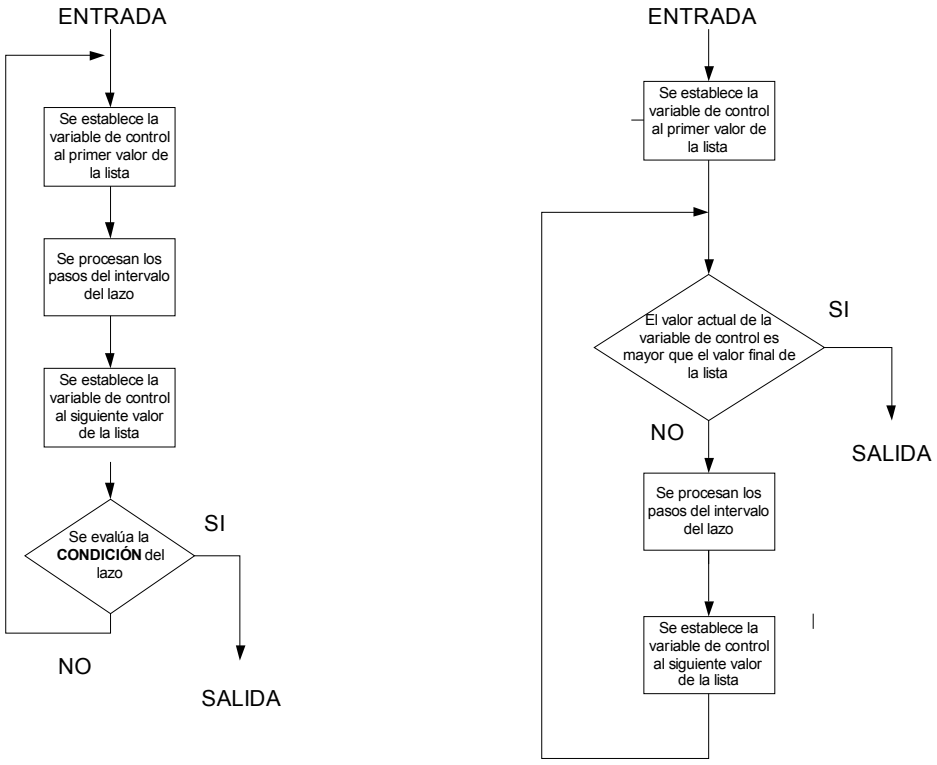


En mucho casos podría desearse disponer de un lazo que se ejecute invariablemente un número determinado de veces, donde el número es conocido. Para esta clase de aplicaciones, conviene una forma modificada de la construcción, que puede expresarse en la siguiente forma.

Repeat "range of loop" **for** "values of loop control variable"

El lazo finaliza cuando el valor final se alcanza o excede, la prueba se lleva acabo al final o antes de la ejecución de los pasos del lazo, como se muestra en la figura 6. En los lazos probados al final los pasos comprendidos en el intervalo del lazo se ejecutan al menos una vez, con independencia de las condiciones de entrada o iniciales. Los lazos probados al inicio permiten la posibilidad de saltar por completo estos pasos, dejando sin ejecutar el lazo, lo que permite una mayor generalidad.

FIGURA 6: Lazos probados al inicio y al final



1.4. Subalgoritmos

Los problemas complejos se enfrentan mejor si se dividen en problemas más pequeños (subproblemas). Esta descomposición es un aspecto importantísimo en la solución de problemas de programación. El subalgoritmo proporciona una herramienta muy poderosa para el desarrollo de algoritmos y programas de computación.

1.4.1. Funciones

El uso de subalgoritmos (en este caso, funciones) altera el flujo del control del algoritmo. Cuando se llama a las funciones, el control pasa a las instrucciones que la definen y, una vez que la función ha sido ejecutada con el argumento dado, el control regresa al punto en que fue llamada por el algoritmo principal, con los valores que han sido calculados con ayuda de la función.

Por razones obvias, ningún sistema puede proporcionar todas las funciones posibles. En muchos problemas se deseará emplear una operación que, por desgracia, no está contenida en el sistema en uso como una función interna. Es importante, también, que el programador tenga la habilidad para definir sus propias funciones.

El formato de una definición de función es:

Función nombre-de-función (lista-de-parámetros): tipo-de-valor-devuelto
Declaraciones e instrucciones
Return variable-que-devuelve-el-valor-al-invocador

1.4.2. Procedimientos

En algunas ocasiones, podría ser deseable especificar una operación que no se especifica en forma conveniente como parte de una expresión. Tales operaciones tienen un campo de aplicación más extenso.

Por estas razones, se introducirá una segunda forma de subalgoritmo que se llamará *procedimiento*. Aunque es similar en muchos aspectos a la función, existen dos diferencias importantes:

1. un procedimiento se llama mediante un enunciado especial, el enunciado *Call*. La ejecución de este enunciado provoca que la ejecución de la rutina que efectuó la llamada sea suspendida momentáneamente y el control pasa a la rutina o procedimiento llamado. Después de que los pasos del procedimiento han sido ejecutados, el control regresa a la rutina que llamó, y continúa con el enunciado inmediatamente siguiente al enunciado Call. La ejecución de la rutina que llamó continúa a partir de este punto.
2. No se regresa un simple valor, como era el caso de una función. Todos lo valores que deben regresarse a la rutina principal lo hacen por medio de una lista de parámetros, pudiendo regresarse cualquier cantidad de ellos.

El formato de definición de un procedimiento es:

Procedimiento nombre-de-procedimiento (lista-de-parámetros)
Declaraciones e instrucciones

El concepto de subalgoritmo es fundamental en programación, y se encuentra disponible en una forma u otra en la mayor parte de los lenguajes de programación, auque la terminología empleada en cada uno de ellos puede variar.

1.4.3. Correspondencia entre argumentos

El método directo de asociar argumentos y parámetros es el llamado paso por valor. Cuando la llamada del procedimiento o de la función se procesa, cada uno de los argumentos es procesado también. Los valores individuales obtenidos son entonces, de hecho, asignados a los respectivos parámetros. Con estos no se hace diferencia entre un argumento que es una variable, una constante o una expresión; todo lo que importa es que el argumento tiene un valor, el cual tienen que transformase en el valor del correspondiente parámetro.

Otro mecanismo para asociar los argumentos con los parámetros es el llamado por paso como variable, más que pasar un valor para cada argumento, el traspaso como variable traslada, de hecho, la variable real misma, la que puede ser potencialmente modificada por la función o el procedimiento llamado.

2. DISEÑO DE PROGRAMAS Y TÉCNICAS DE PROGRAMACIÓN

2.1. Estilo de programación

2.1.1. Nombres

Aún cuando ya existen datos y algoritmos, sólo al recibir un nombre significativo, puede reconocerse y apreciarse debidamente su lugar dentro del programa y adquieren vida independiente.

Para que un programa funcione bien, es de capital importancia conocer exactamente lo que representa cada variable y lo que realiza cada subprograma; los nombres han de escogerse con mucho cuidado de manera que su significado se identifique sin ambigüedades y en forma concisa.

La selección cuidadosa de nombres ayuda mucho a esclarecer un programa y a evitar errores de interpretación y errores comunes. He aquí algunas pautas al respecto:

1) Poner cuidado especial al escoger los nombres de procedimientos, funciones, constantes y todas las variables y tipos empleados en diversas partes del programa. deben ser nombres significativos y sugerir claramente la finalidad del subprograma, variable y cosas afines.
2) Hacer que sean sencillos los nombres de variables que se usan poco y en forma local.
3) Servirse de prefijos o sufijos comunes para asociar los nombres pertenecientes a la misma categoría general.
4) Evitar los errores de ortografía deliberados y los sufijos carentes de significado.

5) Evitar nombres elegantes cuyo significado tiene poca o nula relación el problema

2.1.2. Documentación y formato

El autor de un programa pequeño puede retener en la mente todos los pormenores, por lo cual no necesita la documentación más que para explicar el programa a otra persona. En el caso de programas amplios (y también en el de pequeños después de transcurridos algunos meses), se vuelve imposible recordar cómo cada detalle se relaciona con los demás; por tanto, para escribir programas grandes es indispensable preparar la documentación apropiada a medida que se escribe cada parte del mismo. Un buen hábito consiste en hacerlo a medida que se escriba el programa.

El estilo de la documentación, lo mismo que todo estilo de redacción, es muy personal y muchos estilos diferentes pueden ser igualmente eficaces. Con todo, hay algunas pautas generalmente aceptadas que es preciso respetar:

1) Se pone un prólogo al inicio de cada subprograma que incluye:
    i) Identificación (nombre del programador, fecha, número de versión)
    ii) Declaración del propósito del subprograma y del método aplicado
    iii) Qué cambios introduce el subprograma y que datos usa
    iv) Referencia a otra documentación externa del programa
2) cuando cada variable, constante o tipo se declaren, se explicará lo que es y cómo se emplea, esta información debe ser evidente por el nombre
3) Se introduce cada sección importante del programa por medio de un comentario que señala brevemente su finalidad o acción.
4) No se hacen comentarios que repitan lo que realiza el código
5) También el código deberá explicar cómo funciona el programa. Y la documentación explicará por qué funciona y lo que hace.

Los espacios, líneas en blanco y sangrías en un programa constituyen una forma importante de documentación. Facilitan la lectura del programa, permiten al usuario saber por simple observación qué partes del programa se relacionan entre si, dónde se producen las rupturas principales y cuáles proposiciones están contenidas dentro de cada ciclo o las alternativas de una proposición condicional.

2.1.3. Refinamiento y modularidad.

No son las computadoras sino las personas quienes resuelven los problemas. Por lo regular la parte central del proceso consiste en dividir el problema en otros más pequeños que puedan entenderse con más detalle. Si estos últimos son demasiado difíciles todavía, vuelven a subdividirse y así sucesivamente. Aun cuando un proyecto sea lo suficientemente pequeño para que una sola persona se encargue de él de principio a fin, conviene dividir la tarea: se comienza con una compresión general del problema, se divide en subproblemas y se acomete cada uno a la vez, sin preocuparse por lo demás.

El principio, denominado refinamiento descendente (Top down), es la verdadera clave para escribir programas extensos que funcionen. El principio exige posponer el examen detallado, pero no la precisión ni el rigor. No significa que el programa principal se convierta en una vaga entidad cuya tarea apenas se es descriptible. Por lo contrario, enviará casi todo el trabajo a varios subprogramas (procedimientos y funciones), y a medida que lo escribamos iremos decidiendo cómo se dividirán las tareas entre ellos. Después, al trabajar en un subprograma en particular, antes de empezar sabremos ya lo que habrá de realizar.

A menudo no es fácil decidir exactamente cómo dividir el trabajo en subprogramas y en ocasiones habrá que modificar más tarde la decisión tomada.

Dos pautas nos ayudaran a escoger la forma de dividir el trabajo.

- Cada subprograma deberá ejecutar sólo una tarea.
- Cada subprograma deberá ocultar algo.

2.1. Codificación prueba y refinamiento

La codificación es el proceso de escribir un algoritmo en la sintaxis correcta de un lenguaje de computación; prueba es el proceso consistente en correr el programa en los datos muestra seleccionados para encontrar errores si es que los hay.

2.1.4. Cabos

Luego de codificar el programa principal, la mayor parte de los programadores quieren terminar cuanto antes la escritura y codificación de los subprogramas, a fin de comprobar si funciona el proyecto en su totalidad. En un proyecto pequeño este enfoque dará resultado; pero en otros más amplios escribir y codificar todos los subprogramas será una tarea tan enorme, que cuando se termine se habrán olvidado ya mucho de los detalles del programa principal y de los subprogramas escritos antes. Es mucho más fácil entender y depurar un programa cuando está fresco en nuestra mente. De ahí que, tratándose de proyectos extensos, sea preferible depurar y probar cada subprograma en cuanto esté terminado a esperar hasta que el proyecto haya sido codificado por completo.

Para compilar correctamente el programa, debe haber algo en el lugar de cada subprograma que se usa; por eso debemos usar con este fin subprogramas sustitutos llamados **cabos**.

Los más sencillos son los que no hacen nada en absoluto.

```
procedure inicializa; begin end;
procedure EscribeMapa; begin end;
function CuentaVecino(i: renglón; j: columna):integer; begin end;
```

2.1.5. Entrada y salida

En los programas diseñados para varios usuarios, los procedimientos que realizan la entrada y salida suelen ser los más largos. La entrada del programa ha de verificarse con sumo cuidado y es preciso asegurarse de que sea válida y congruente; los errores en la entrada se procesarán en formas tales que eviten un fracaso catastrófico o la producción de resultados ridículos. La salida se organiza y se formatea, teniendo muy en cuenta lo que debe o no imprimirse y atendiendo además a las diversas alternativas acordes a las circunstancias. Un buen hábito consiste en hacer la entrada y salida sean módulos aparte, para que sea fácil cambiarlas y puedan adaptarse a su sistema.

2.1.6. Manejadores

En proyectos pequeños, cada subprograma se inserta en su sitio correspondiente tan pronto haya sido escrito, y el programa resultante se depura y se prueba en lo posible. Tratándose de proyectos vastos, la compilación del proyecto entero puede obstaculizar la verificación de un nuevo subprograma que va a ser depurado, siendo difícil saber si un subprograma está funcionando bien o no.

Una manera de depurar y probar un solo programa consiste en escribir un programa auxiliar breve, cuya finalidad es proporcionar la entrada necesaria para el subprograma, llamarlo y evaluar el resultado. A ese programa auxiliar se le llama **manejador** del subprograma. Al utilizar manejadores, es posible aislar cada subprograma y estudiarlo por separado; de ese modo se detectan rápidamente los errores.

2.1.7. Principios de prueba de programas

Hasta ahora nada hemos dicho sobre la selección de los datos con que se probarán los programas y subprogramas. Esta selección depende estrechamente del proyecto en desarrollo, por lo cual la calidad de los datos de prueba es más importante que la cantidad.

A la mayor parte de los usuarios de programas extensos no les interesa los detalles de su funcionamiento; lo único que desean es conseguir respuestas. Es decir, quieren tratar el programa como una **caja negra**. De manera análoga, los datos de prueba se escogerán atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

A continuación se citan os criterios mínimos que nos guiarán al escoger los datos de prueba:

1) **Valores fáciles.** El programa se depurará con datos de fácil comprobabilidad.
2) **Valores típicos realistas.** Siempre se ensayará un programa con datos seleccionados para que representen cómo se aplicará. Tales datos han de ser suficientemente sencillos, de modo que los resultados sean verificables en forma manual.
3) **Valores extremos.** Muchos programas cometen errores en los límites de su rango e aplicaciones.

4) **Valores ilegales.** “Basura entra, basura sale”, cuando en un programa entra basura, su salida habrá de ser por lo menos un mensaje de error adecuado. Es preferible que el programa ofrezca alguna indicación de probables errores en la entrada y que realice cálculos que sigan siendo factibles luego de desechar la entrada equivocada.

El segundo enfoque en la selección de los datos de prueba principia con la observación de que un programa difícilmente puede considerarse como probado por completo si su código contiene partes que nunca han sido ejecutadas.

En el método de la **caja de cristal**, se analiza la escritura lógica del programa y, para cada alternativa que pueda presentarse, los datos de prueba ideados conducirán a ella. Así pues, se procura escoger los que verifiquen cada posibilidad en las proposiciones **case**, las cláusulas de cada proposición **if** y la condición de terminación de cada ciclo.

En el caso de un programa extenso, este método es sin duda impráctico, pero en un módulo pequeño constituye un excelente medio de prueba y depuración. En un programa bien diseñado, cada módulo incluirá unos pocos ciclos y opciones. De ahí que basten algunos casos de prueba bien seleccionados para probar cada módulo por separado. Por tanto, un buen criterio de prueba para proyectos extensos consiste en aplicar los métodos de la caja de cristal a cada módulo pequeño conforme se escriba; luego se usan esos datos en las secciones más amplias del programa una vez terminadas.

Un enfoque más en la prueba de programas, enfoque que lamentablemente se usa mucho, podríamos llamarlo método de la **caja de Pandora**. Cosiste en abstenerse de realizar pruebas luego de depurar bastante bien un proyecto; se deja al cliente que lo ensaye y acepte. Sobra decir que el resultado es una bomba de tiempo.

2.2. Mantenimiento de un programa

El mantenimiento del programa, es decir, el ajuste de los programas que existen para reunir requerimientos siempre cambiantes, consume gran parte del tiempo del trabajo de los programadores; es muy común, de hecho, que se emplee más tiempo en el mantenimiento de un programa que en su desarrollo original.

El mantenimiento de los programas no se refiere a la reparación o cambio de partes deterioradas, sino a las modificaciones que deben hacerse a los defectos del diseño, lo cual puede incluir el desarrollo de funciones adicionales para reunir nuevas necesidades.

Sin duda es importante que un programa trabaje correctamente y que sea confiable, es decir, que reúna todos los requisitos exigidos y que los errores inesperados ocurran muy rara vez.

Los programas parecen necesitar un procesamiento continuo de mantenimiento y modificación para mantenerse al día con los requerimientos cambiantes de la tecnología y con la instalación de ellos en las maquinas. Las capacidades de modificación y mantenimiento son características esenciales de los programas reales. Que un programa sea fácil de leer y de comprender son prerrequisitos importantes para lograr su mantenimiento y modificación apropiados.

En resumen, se desean programas correctos, confiables, fáciles de mantener, modificables, legibles y comprensibles.

Casi todos los programadores pasan 90% del tiempo realizando 10% de las instrucciones. Encuentre ese 10% y concéntrese en mejorar allí la eficiencia.

3. AUTOMATIZACIÓN ESTRUCTURADA DE LA PRÁCTICA DE DESTILACIÓN DE UNA SOLUCIÓN BINARIA

La automatización es un sistema en el cual se trasfieren tareas de producción, realizadas habitualmente por operadores humanos, a un conjunto de elementos tecnológicos.

Un sistema automatizado consta de dos partes principales:

- Parte de mando
- Parte operativa

La *Parte Operativa* es la parte que actúa directamente sobre la máquina. Son los elementos que hacen que la máquina se mueva y realice la operación deseada. Los elementos que forman la parte operativa son los accionadores de las máquinas como motores, cilindros, compresores... y los captadores como fotodiodos, finales de carrera, etc.

La *Parte de Mando* suele ser un autómata programable (tecnología programada), aunque hasta muy recientemente se utilizaban relés electromagnéticos, tarjetas electrónicas o módulos lógicos neumáticos (tecnología cableada). En un sistema de fabricación automatizado, el autómata programable está en el centro del sistema. Éste debe ser capaz de comunicarse con todos los componentes del sistema automatizado.

Un autómata programable industrial (API) o programable logic controller (PLC), es un equipo electrónico, programable en lenguaje no informático, diseñado para controlar en tiempo real y en ambiente de tipo industrial, procesos secuenciales.

Un PLC trabaja en base a la información recibida por los captadores y el programa lógico interno, actuando sobre los accionadores de la instalación.

El PLC por sus especiales características de diseño tiene un campo de aplicación muy extenso. La constante evolución del hardware y software amplía constantemente este campo para poder satisfacer las necesidades que se detectan en el espectro de sus posibilidades reales.

Su utilización se da fundamentalmente en aquellas instalaciones en donde es necesario un proceso de maniobra, control, señalización, etc., por tanto, su aplicación abarca desde procesos de fabricación industriales de cualquier tipo a transformaciones industriales, control de instalaciones, etc.

3.1. Inicio del diseño

Las técnicas de diseño mencionadas en los capítulos anteriores, servirán de referencia para el diseño automatizado de la práctica de destililación de una solución binaria.

Requerimientos del usuario:

1. Fácil de operar
2. Determinar cuando se finaliza el proceso de destilación
3. Detener el proceso de destilación cuando se presente el limite de temperatura superior

4. Permitir fácil acceso para:
   4.1.El inicio o fin del proceso de destilación
   4.2.Aumentar o disminuir la potencia por medio de resistencias que calentarán el calderín
   4.3.La toma de datos de temperatura en cada plato de la torre.
5. Permitir el reinicio del proceso de destilación después de la toma de muestras y datos

Especificaciones funcionales:

- *Sensores de temperatura*: pueden ser usadas para la toma y control de temperatura en los platos de la torre.
- *Relés*: para la correcta conmutación de las resistencias en el calentador del calderín para aumentar o disminuir la potencia requerida para la realización de la práctica.
- *Ejecutar un programa con interfase visual, desde una computadora con Windows como sistema operativo (HMI)*: para que el intercambio de información entre el autómata programable y la HMI facilite a los estudiantes el inicio y finalización de la práctica de destilación de una solución binaria.

A) Entrada
   a) HMI
      i) Mouse
      ii) Teclado
      iii) Puerto de comunicación
   b) Autómata
      i) Detectores de temperatura
      ii) Puerto de comunicación
B) Salida
   a) HMI

i) Puerto de comunicación

b) Autómata

i) Relés

ii) Puerto de comunicación

C) Funciones

a) Los relés se activarán al inicio del proceso de destilación para conmutar las resistencias del calentador del calderín y se mantendrán activos hasta que se estabilice la temperatura en cada plato de la torre

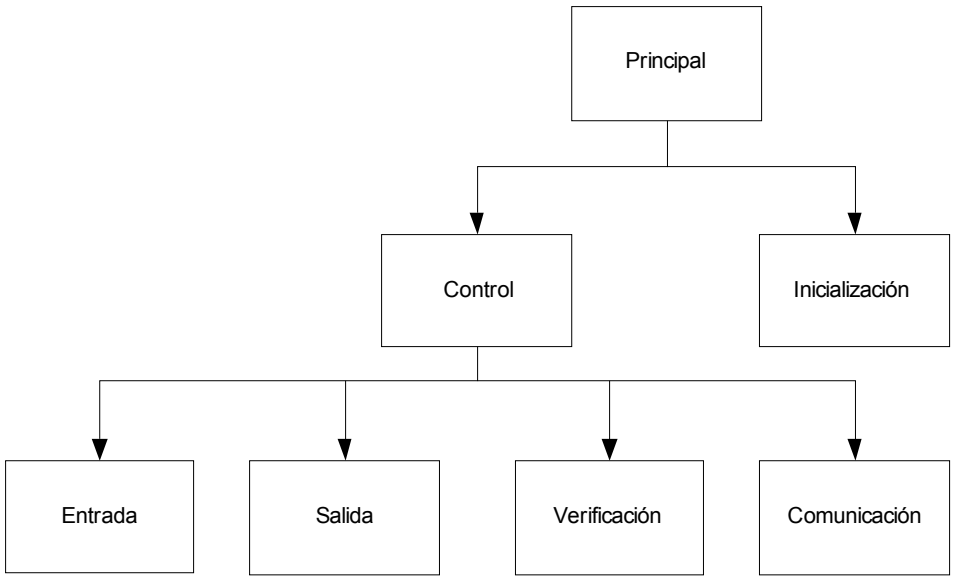
b) Los puertos de comunicación se utilizarán para el control y transferencia de datos entre el autómata y la interfase visual (HMI.)

Modularización funcional:

- *Módulo principal*: se encarga de administrar las funciones de cada procedimiento en la realización del proceso de destilación
- *Módulo de entrada*: se encarga de todas las entradas necesarias para llevar acabo el proceso de destilación
- *Módulo de salida*: se encarga de todas las salidas necesarias para llevar acabo el proceso de destilación
- *Módulo de verificación*: se encarga de obtener de los datos recibidos de los sensores de temperatura en cada plato de la torre, cuando ha de finalizarse el proceso de destilación.
- *Módulo de comunicación*: se encargara del flujo de información entre el autómata y la interfase visual (HMI)
- *Módulo de control*: se encargará del control de los datos de entrada/salida entre el autómata y la interfase visual (HMI)
- *Módulo de inicialización*: establecerá todas la variables a utilizar en el proceso a su valores iniciales para una correcta funcionalidad en el desarrollo de cada módulo

La estructura modular del diseño puede observarse en la figura 7.

FIGURA 7: Refinamiento descendente: Top down



3.2. Software

3.2.1. Interfase visual (HMI)

Interfase visual es un programa que tiene como tarea el intercambio de información entre el autómata y la computadora personal, actuando o simulando una HMI.

El algoritmo general para diseñar la Interfase visual consta de 8 pasos.

1. Inicializar variables de estado y de control
2. Ejecutar rutinas visuales de selección e ingreso de datos por el usuario
3. Seleccionar el puerto de comunicación a usar

4. Abrir el puerto
5. Habilitar opciones de usuario de inicio y finalización del proceso

   5.1.Si se inicia el módulo de inicio de proceso deshabilitar opciones de ingreso de datos por el usuario y habilitar opciones de envío de datos al autómata y finalización del proceso de destilación.
6. Iniciar ciclo de lectura de datos recibidos por el autómata y mostrarlos en pantalla
7. Verificar fin de proceso y de control de temperatura para no sobrepasar el límite superior de temperatura ingresada por el usuario.

   7.1.Si el fin de proceso es verificado, detener todo proceso de envío de información entre el autómata y la HMI y habilitar opciones de ingreso de datos por el usuario.
8. Esperar opciones de usuario de finalización del programa o reiniciar otra practica.

El diagrama modular del diseño puede observarse en la figura 8, y el diseño final en las figuras 9 y 10.

FIGURA 8: Diseño modular del programa visual

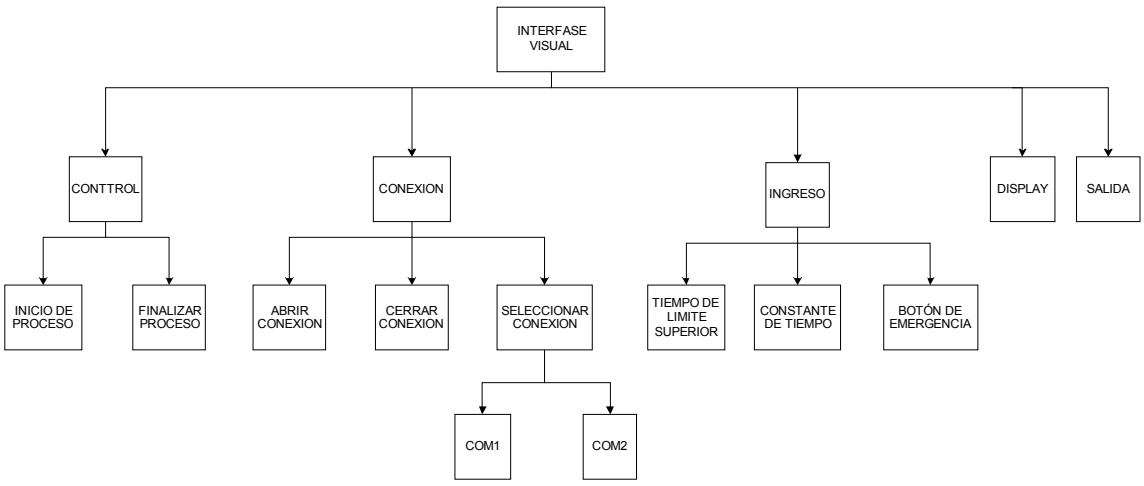


FIGURA 9. Opciones de usuario habilitadas

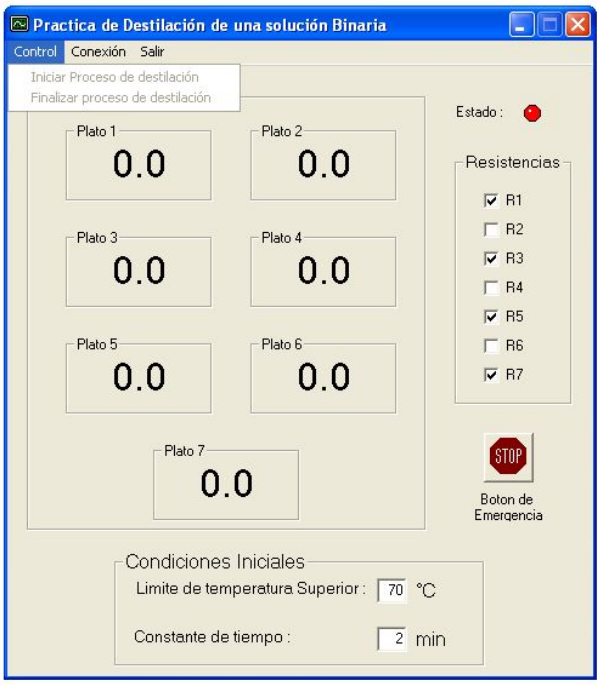
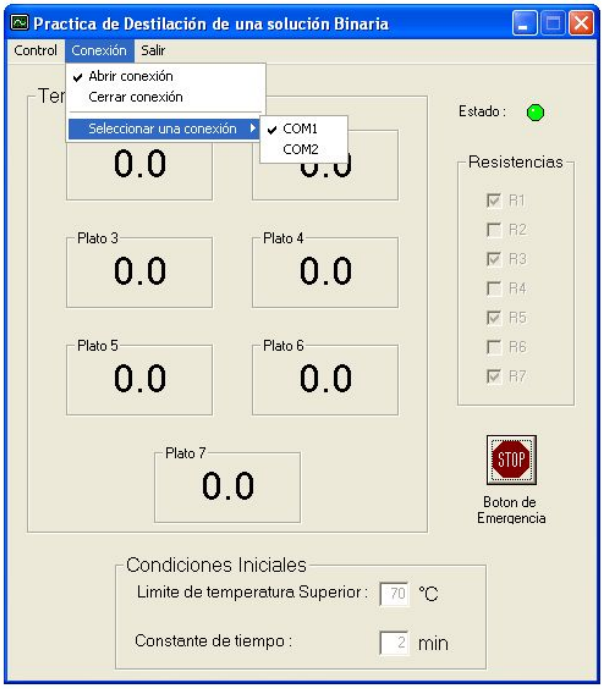


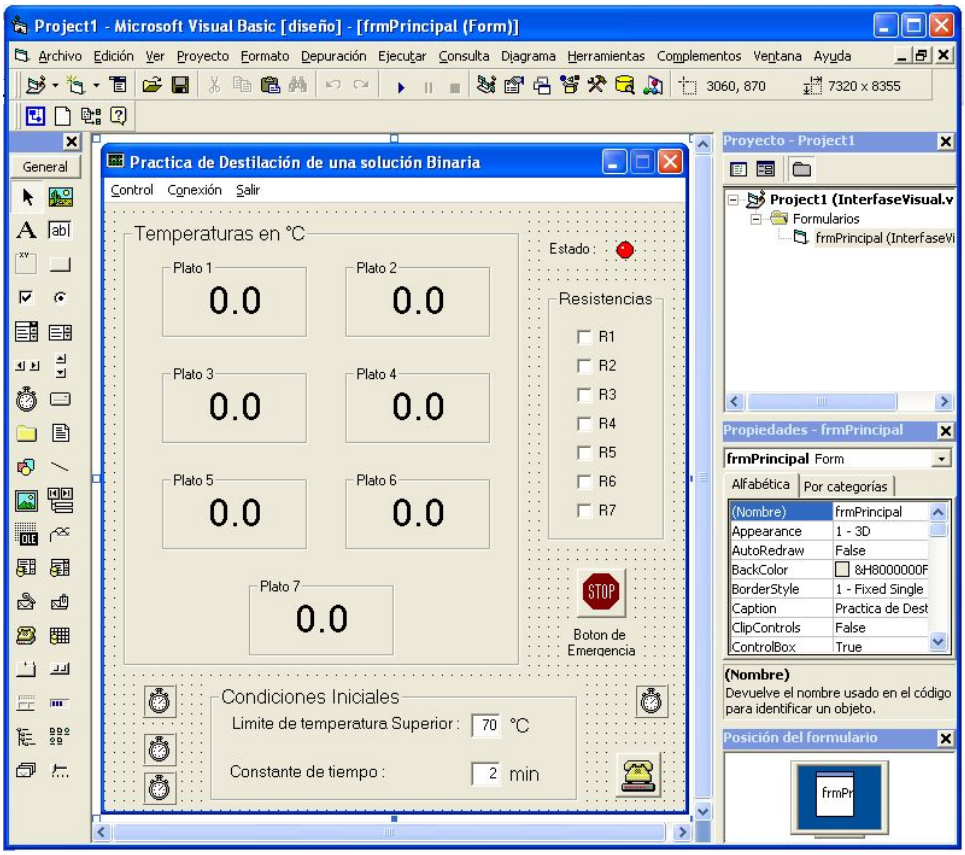
FIGURA 10: Opciones de usuario deshabilitadas y proceso activo



Para dar una mejor comprensión se realizará la conversión de la lógica del algoritmo (diagrama de flujo) a lenguaje visual.

En la figura 11, se observa el ambiente general del lenguaje de alto nivel Visual Basic, lenguaje que se utilizo para implementar el diseño de la interfase visual, se puede observar la ubicación final de cada procedimiento que se ejecutará en la ventana principal o módulo principal llamada *frmPrincipal* (interfase visual en el diseño modular) de donde se ejecutarán todos los módulos restantes.

FIGURA 11: Diseño de la interfase visual en Visual Basic

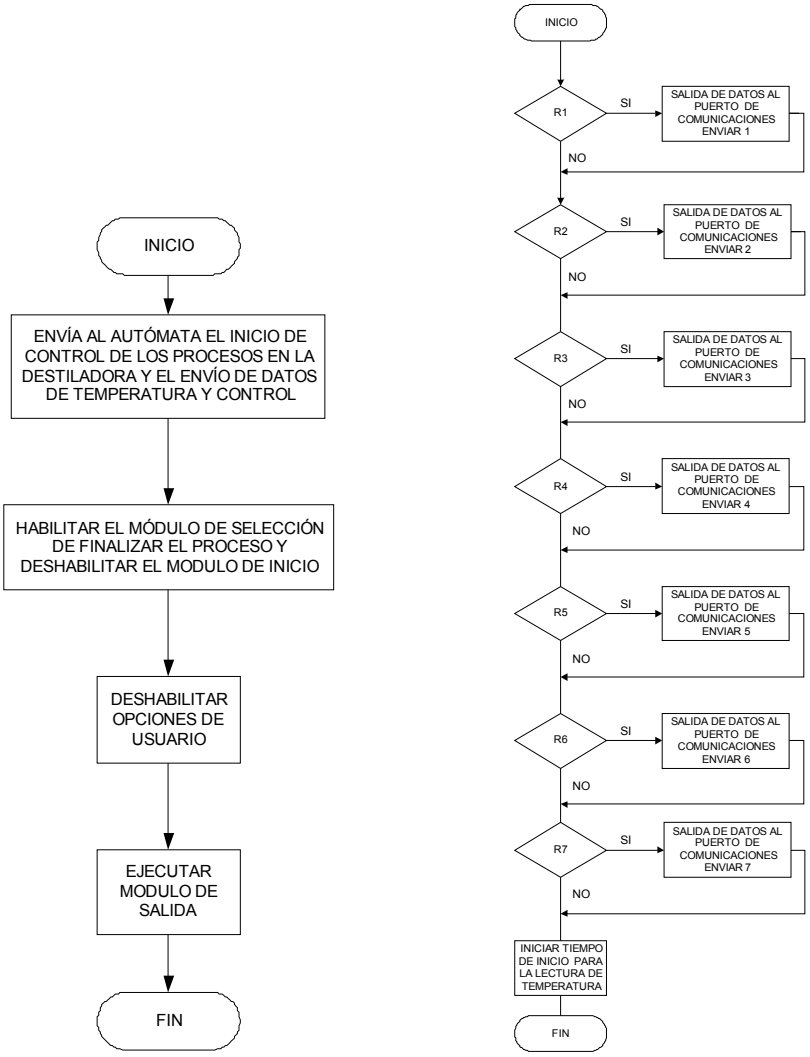


Módulo interfase visual: Código fuente inicial.

```
'--------------------------------------------------
' Interfasevisual es un programa que intercambia
' información con un autómata programable S7-200
' CPU 224 de Siemens.
'
' Copyright (c) 2005, v1.0
' por Walter Giovanni Alvarez Marroquin.
'
' Facultad de Ingeniería USAC.
'--------------------------------------------------

Option Explicit
' Declaracion de variables globales a ser usadas en el programa


Dim ret, NewPort As Integer
Dim plato, segundos As Integer
Dim platoA, platoB, platoC, platoD, platoE, platoF, PlatoG, p As Double


Private Sub Form_Load()
' Establece los estados de las variables a sus valores de inicio
' cuando se ejecuta por primera vez el programa.

  openport.Enabled = False
  closeport.Enabled = False

  inicio.Enabled = False
  finalizar.Enabled = False

  'Configuracion del puerto
  Comm1.Settings = "9600,n,8,1"

  closeport.Checked = Not (Comm1.PortOpen)
  image1.Visible = True

  platoA = 1
  platoB = 1
  platoC = 1
  platoD = 1
  platoE = 1
  platoF = 1
  PlatoG = 1

End Sub
```

Módulo de control:

FIGURA 12: Diagrama de flujo de los módulos INICIO Y SALIDA



En la figura 12 se muestra el diseño de la lógica de algoritmo de la rutina o procedimiento inicio de proceso y salida, el código fuente en lenguaje de alto nivel:

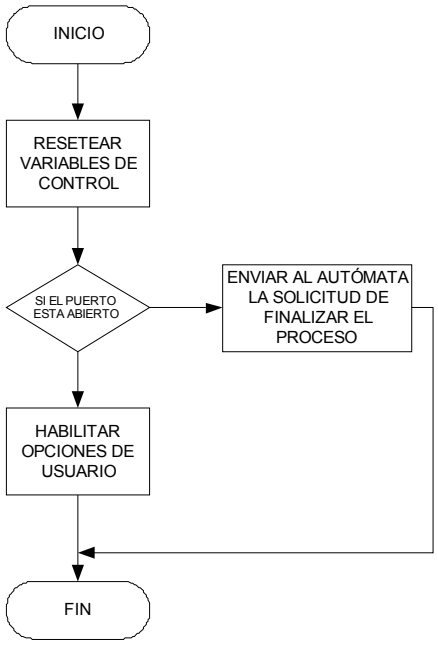
```
Private Sub inicio_Click()
' Módulo o Rutina de inicio de proceso de destilación.
   segundos = 0

   ' Confirma al autómata el inicio de control y envío de datos.
   Comm1.Output = "I"
    image1.Visible = False
   image2.Visible = True

   ' Habilita el módulo de selección para Finalizar el proceso.
   finalizar.Enabled = True
   inicio.Enabled = False

   ' Deshabilitar opciones de usuario.
   LimiteSuperior.Enabled = False
   ctetiempo.Enabled = False
   Abortar.Enabled = True
   R1.Enabled = False
   R2.Enabled = False
   R3.Enabled = False
   R4.Enabled = False
   R5.Enabled = False
   R6.Enabled = False
   R7.Enabled = False

   ' Llama la rutina o módulo de salida.
   Salidas

End Sub
```

Y para el módulo de salida:

```
Private Sub Salidas()
' Envía al autómata las resistencias que
' serán conmutadas en el calentador del calderín.

   If R1.Value = 1 Then
      Comm1.Output = "1"
   End If
   If R2.Value = 1 Then
      Comm1.Output = "2"
   End If
   If R3.Value = 1 Then
      Comm1.Output = "3"
   End If
   If R4.Value = 1 Then
      Comm1.Output = "4"
   End If
   If R5.Value = 1 Then
      Comm1.Output = "5"
   End If
   If R6.Value = 1 Then
      Comm1.Output = "6"
   End If
   If R7.Value = 1 Then
      Comm1.Output = "7"
   End If
```

```
    ' Inicia el ciclo de tiempo para la toma y lectura de las
    ' temperaturas en cada plato de la torre.
     TiempodeInicio.Enabled = True

End Sub
```

FIGURA 13: Diagrama de flujo del procedimiento FINALIZAR PROCESO



En la figura 13 se muestra el diseño de la lógica de algoritmo de la rutina o procedimiento finalizar proceso, el código fuente en lenguaje de alto nivel:

```
Private Sub finalizar_Click()
' Finaliza todo proceso en la destiladora

    ' Deshabilitar los tiemers de control
    TiempodeInicio.Enabled = False
    TiempodeLectura.Enabled = False
    CtrlProceso.Enabled = False
    reloj.Enabled = False

    ' Enviar al autómata la solicitud de finalización de procesos.
    If Comm1.PortOpen Then
       Comm1.Output = "F"
    End If
```

```
' Inicializa toda variable a su estado de inicio
' y habilita opciones de usuario.

Abortar.Enabled = False
image1.Visible = True
image2.Visible = False
LimiteSuperior.Enabled = True
ctetiempo.Enabled = True
R1.Enabled = True
R2.Enabled = True
R3.Enabled = True
R4.Enabled = True
R5.Enabled = True
R6.Enabled = True
R7.Enabled = True
inicio.Enabled = True
finalizar.Enabled = False

End Sub
```

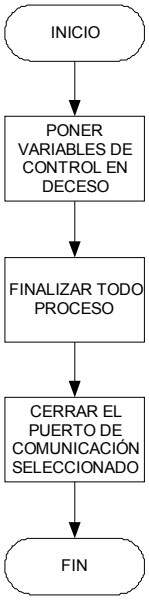
Módulo de conexión:

FIGURA 14: Diagrama de flujo del procedimiento ABRIR CONEXIÓN

En la figura 14 se muestra el diseño de la lógica de algoritmo de la rutina o procedimiento abrir conexión, el código fuente en lenguaje de alto nivel:

```
Private Sub openport_Click()
' Módulo o rutina para la habilitación del puerto de comunicación a usar.

   On Error Resume Next
   Comm1.CommPort = NewPort
   If Err Then
      MsgBox Error$, 48
      Exit Sub
   End If

   'configura el puerto para recibir todos los bytes
   Comm1.InputLen = 0

   On Error Resume Next
   Comm1.PortOpen = True


  If Err Then
      MsgBox Error$, 48
      Exit Sub
   End If

   ' Habilita opciones del usuario.
   inicio.Enabled = True
   openport.Checked = Comm1.PortOpen
   closeport.Checked = Not (Comm1.PortOpen)

End Sub
```

FIGURA 15: Diagrama de flujo del procedimiento CERRAR CONEXIÓN



En la figura 15 se muestra el diseño de la lógica de algoritmo de la rutina o procedimiento cerrar conexión, el código fuente en lenguaje de alto nivel:

```
Private Sub closeport_Click()
'Cierra el puerto de comunicación seleccionado y finaliza todo proceso.

   TiempodeLectura.Enabled = False
   inicio.Enabled = False
   finalizar_Click
   finalizar.Enabled = False

   On Error Resume Next
      Comm1.PortOpen = False
      openport.Checked = Comm1.PortOpen
      closeport.Checked = Not (Comm1.PortOpen)

   If Err Then
      MsgBox Error$, 48
      Exit Sub
   End If
End Sub
```

El código fuente para la selección del puerto de comunicación serial a usar:

```
Private Sub puerto1_Click()
' Seleccionar el puerto serial 1

   NewPort = 1
   puerto1.Checked = True
   puerto2.Checked = False
   openport.Enabled = True
   closeport.Enabled = True

End Sub

Private Sub puerto2_Click()
' Seleccionar el puerto serial 2

   NewPort = 2
   puerto1.Checked = False
   puerto2.Checked = True
   openport.Enabled = True
   closeport.Enabled = True

End Sub
```

Módulo de ingreso:

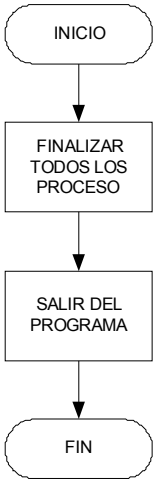
Los datos son ingresados por el usuario cuando el programa se esta ejecutando, esto se puede observar en la figura 9.

Módulo de display:

Los datos de temperatura recibidos por el autómata se muestran en pantalla por medio de lugares específicos para cada plato de la torre en la destiladora como se puede observar en las figuras 9 y 10.

Módulo de salida:

FIGURA 16: Diagrama de flujo del procedimiento SALIDA.



En la figura 16 se muestra el diseño de la lógica de algoritmo de la rutina o procedimiento de salida, el código fuente en lenguaje de alto nivel:

```
Private Sub salir_Click()
'Módulo para finalizar los programas en el autómata y PC.

   ' Llama la rutina o módulo de finalizar
   finalizar_Click

   ' Termina la ejecución del programa
   End

End Sub
```

Funciones temporizadas:

FIGURA 17: Diagrama de flujo del timer TIEMPO DE INICIO

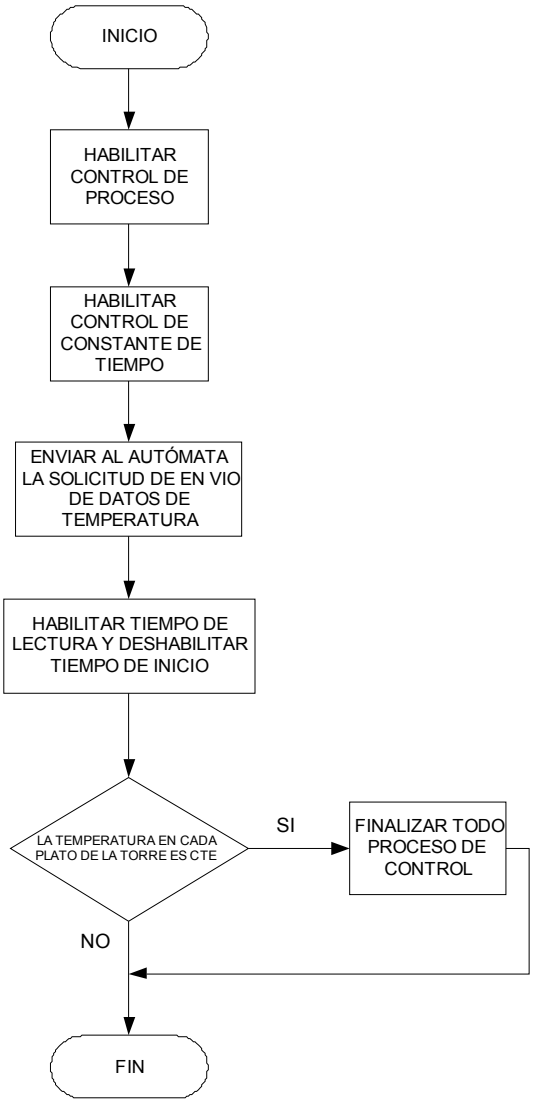
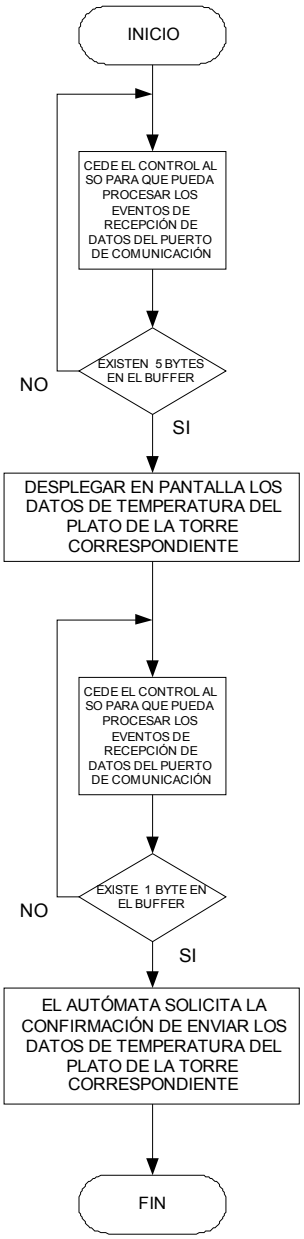


FIGURA 18: Diagrama de flujo del timer TIEMPO DE LECTURA



En la figura 17 y 18 se muestra el diseño de la lógica de algoritmo de las funciones temporizadas tiempo de inicio y tiempo de lectura, el timer de inicio se ejecutará después de un minuto y el timer de lectura se ejecutará después de un segundo, el código fuente en lenguaje de alto nivel:

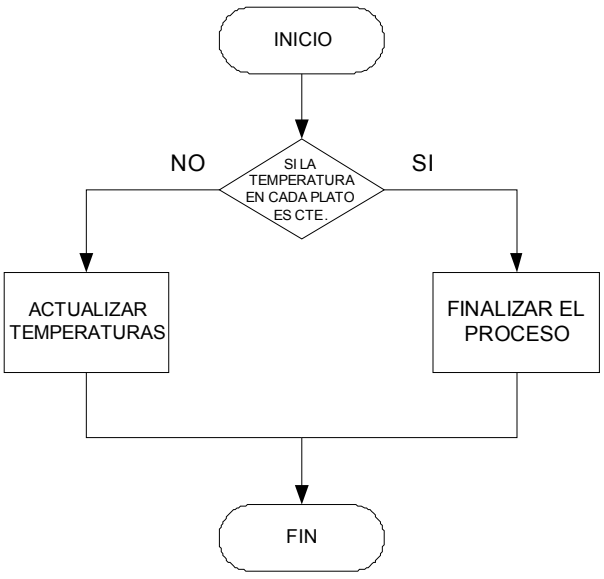
```
Private Sub TiempodeInicio_Timer()
' Timer de inicio de lectura de datos de temperatura.

   ' Tiempo seleccionado por el usuario.

   reloj.Enabled = True

   ' Inicio de envío de datos desde el autómata y la interfase visual
   ' empezando por el primer plato de la torre.
   Comm1.Output = "A"
   plato = 1
   TiempodeLectura.Enabled = True
   TiempodeInicio.Enabled = False

   ' Finalizar todo proceso si la temperatura sobrepasa el límite.
   If (plato1 Or plato2 Or plato3 Or plato4 Or plato5 Or plato6 Or plato7) >= LimiteSuperior.Text Then
      finalizar_Click
   End If

End Sub
```

Y para el módulo de tiempo de lectura

```
Private Sub TiempodeLectura_Timer()
' Establece cuando el programa intercambiará información con el autómata
' para mostrar las lecturas de temperatura en cada plato de la torre.

' Espera que los 5 bytes estén en el buffer de datos.
' Información de temperatura para el plato correspondiente.
Do
   DoEvents
Loop Until Comm1.InBufferCount = 5

   Select Case plato

      Case 1
         plato1 = Comm1.Input

      Case 2
         plato2 = Comm1.Input

      Case 3
         plato3 = Comm1.Input

      Case 4
         plato4 = Comm1.Input

      Case 5
         plato5 = Comm1.Input

      Case 6
         plato6 = Comm1.Input

      Case 7
         plato7 = Comm1.Input
   End Select
```

```
' Espera un byte en el buffer de datos.
' Confirma al autómata la solicitud de enviar
' los datos de temperatura del plato correspondiente.

Do
   DoEvents
Loop Until Comm1.InBufferCount = 1

   Select Case Comm1.Input
      Case "1"
         TiempodeInicio.Enabled = True
         TiempodeLectura.Enabled = False

      Case "2"
         plato = 2
         Comm1.Output = "B"
      Case "3"
         plato = 3
         Comm1.Output = "C"
      Case "4"
         plato = 4
         Comm1.Output = "D"
      Case "5"
         plato = 5
         Comm1.Output = "E"
      Case "6"
         plato = 6
         Comm1.Output = "F"
      Case "7"
         plato = 7
         Comm1.Output = "G"
   End Select

End Sub
```

FIGURA 19: Diagrama de flujo del timer CONTROL DE PROCESO



En la figura 19 se muestra el diseño de la lógica de algoritmo de la función temporizada control de proceso, este timer se ejecutará después de un segundo, el código fuente en lenguaje de alto nivel:

```
Private Sub CtrlProceso_Timer()
' Controla cuando el proceso de destilación ha de terminarse.

If platoA = plato1 And platoB = plato2 And platoC = plato3 And platoD = plato4 And platoE = plato5 And platoF =
plato6 And              PlatoG = plato7 Then
   finalizar_Click
Else
' Actualiza los valores de temperatura cada vez que se habilita el timer.

   platoA = plato1
   platoB = plato2
   platoC = plato3
   platoD = plato4
   platoE = plato5
   platoF = plato6
   PlatoG = plato7
End If

End Sub
```

3.2.2. Ejecución de procesos (PLC)

El programa que se ejecutará en el autómata tiene como tarea el control de todo el proceso que se realizará en la destiladora.

El algoritmo general para diseñar el programa lógico interno consta de 5 pasos:

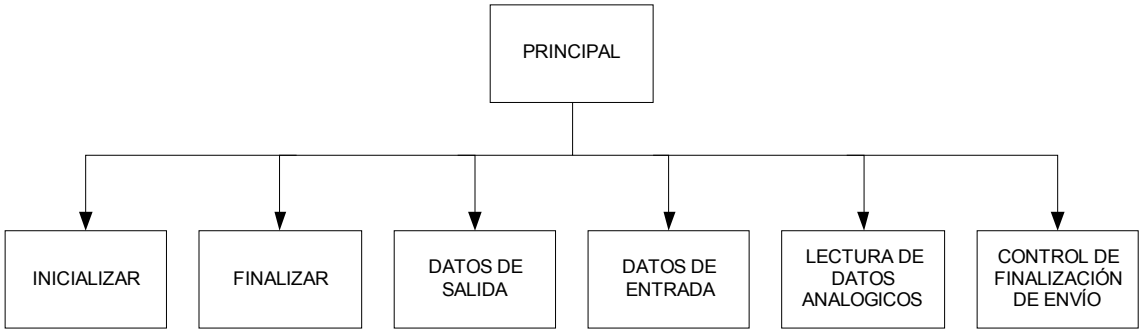
1. Activar el módulo de comunicación del autómata
2. Esperar la confirmación de la interfase visual, para iniciar con la inicialización del Hardware y esperar el envío de las opciones del usuario para la conmutación de las resistencias del calderín.
3. Iniciar ciclo de lectura de los módulos analógicos
4. Esperar recibir solicitud de envío de datos de temperatura.

4.1.Si la solicitud es confirmada entonces deshabilitar la lectura de los datos de los módulos analógicos y habilitar las funciones temporizadas

5. Si la secuencia de envío es correcta entonces repetir los pasos 3 y 4 hasta que se confirme la finalización del proceso desde la interfase visual habilitando el módulo de finalizar.

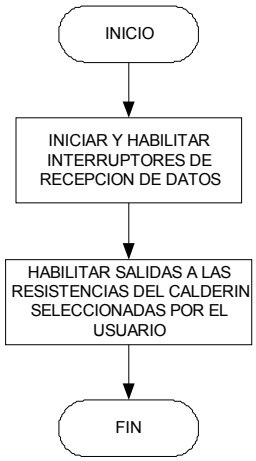
El diagrama modular del diseño puede observarse en la figura 20.

FIGURA 20: Diseño modular del programa lógico interno



Módulo principal:

FIGURA 21: Diagrama de flujo del módulo PRINCIPAL.



En la figura 21 se muestra el diseño de la lógica de algoritmo del módulo principal, el código fuente en lenguaje de alto nivel:

```
Network 1 // Inicializa y activa el puerto de comunicación del autómata.

LD    SM0.1
MOVB  8, SMB30

ATCH  INDATA, 8
ENI

Network 2 // Habilita la salida a la resistencia 1 del calentador del calderín.
LD    M1.1
=     Q0.1

Network 3 // Habilita la salida a la resistencia 2 del calentador del calderín.
LD    M1.2
=     Q0.2

Network 4 // Habilita la salida a la resistencia 3 del calentador del calderín.
LD    M1.3
=     Q0.3

Network 5 // Habilita la salida a la resistencia 4 del calentador del calderín.
LD    M1.4
=     Q0.4

Network 6 // Habilita la salida a la resistencia 5 del calentador del calderín.
LD    M1.5
=     Q0.5

Network 7 // Habilita la salida a la resistencia 6  del calentador del calderín.
LD    M1.6
=     Q0.6

Network 8 // Habilita la salida a la resistencia 7del calentador del calderín.
LD    M1.7
=     Q0.7
```

Módulo inicializar:

FIGURA 22: Diagrama de flujo del módulo INICIALIZAR

En la figura 22 se muestra el diseño de la lógica de algoritmo del módulo de inicialización, el código fuente en lenguaje de alto nivel:

Network 1 // Inicialización de las marcas de control que habilitan las resistencias del calentador del calderín

```
LD    SM0.0
R     M1.1, 1
R     M1.2, 1
R     M1.3, 1
R     M1.4, 1
R     M1.5, 1
R     M1.6, 1
R     M1.7, 1
```

Network 2 // Inicialización de las marcas de control que habilitan las funciones de envió de información.

```
LD    SM0.0
R     M3.1, 1
R     M3.2, 1
R     M3.3, 1
R     M3.4, 1
R     M3.5, 1
R     M3.6, 1
R     M3.7, 1
```

Network 3 // Inicialización de las variables de configuración y control de interrupciones

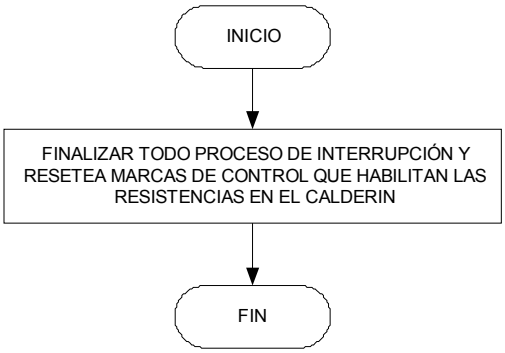
```
LD    SM0.0
MOVB  200, SMB34
MOVB  10, SMB35
ATCH  LECTURA, 10
```

Network 4 // Inicialización de datos de tablas de envió al puerto de comunicación cuando ha terminado el envió de los datos de temperatura a la interfase visual.

```
LD    SM0.0
MOVB  1, VB0
MOVB  1, VB1
MOVB  1, VB2
MOVB  2, VB3
MOVB  1, VB4
MOVB  3, VB5
MOVB  1, VB6
MOVB  4, VB7
MOVB  1, VB8
MOVB  5, VB9
MOVB  1, VB10
MOVB  6, VB11
MOVB  1, VB12
MOVB  7, VB13
```

Módulo finalizar:

FIGURA 23: Diagrama de flujo del módulo FINALIZAR



En la figura 23 se muestra el diseño de la lógica de algoritmo del módulo de finalización, el código fuente en lenguaje de alto nivel:

Network 1 // Finaliza todo proceso de interrupción y resetea las marcas de control que habilitan las resistencias en el alentador del calderín.

```
LD    SM0.0
R     M1.1, 1
R     M1.2, 1
R     M1.3, 1
R     M1.4, 1
R     M1.5, 1
R     M1.6, 1
R     M1.7, 1
R     M3.1, 1
R     M3.2, 1
R     M3.3, 1
R     M3.4, 1
R     M3.5, 1
R     M3.6, 1
R     M3.7, 1
DTCH  9
DTCH  10
DTCH  11
```

Módulo salida de datos:

FIGURA 24: Diagrama de flujo del módulo SALIDA DE DATOS

En la figura 24 se muestra el diseño de la lógica de algoritmo del módulo de salida de datos, el código fuente en lenguaje de alto nivel:

```
Network 1 // Desasociar interrupción de lectura de datos
LD     SM0.0
DTCH   11

Network 2 // Habilita el buffer de datos plato1 y los envía a la interfase visual
LD     M3.1
MOVB   5, VB110
ITA    VW100, VB111, 81
XMT    VB110, 0

Network 3 // Habilita el buffer de datos plato2 y los envía a la interfase visual
LD     M3.2
MOVB   5, VB210
ITA    VW200, VB211, 81
XMT    VB210, 0
```

Network 4 // Habilita el buffer de datos plato3 y los envía a la interfase visual
LD M3.3
MOVB 5, VB310
ITA VW300, VB311, 81
XMT VB310, 0

Network 5 // Habilita el buffer de datos plato4 y los envía a la interfase visual
LD M3.4
MOVB 5, VB410
ITA VW400, VB411, 81
XMT VB410, 0

Network 6 // Habilita el buffer de datos plato5 y los envía a la interfase visual
LD M3.5
MOVB 5, VB510
ITA VW500, VB511, 81
XMT VB510, 0

Network 7 // Habilita el buffer de datos plato6 y los envía a la interfase visual
LD M3.6
MOVB 5, VB610
ITA VW600, VB611, 81
XMT VB610, 0

Network 8 // Habilita el buffer de datos plato7 y los envía a la interfase visual
LD M3.7
MOVB 5, VB710
ITA VW700, VB711, 81
XMT VB710, 0

Módulo de lectura de datos analógicos:

FIGURA 25: Diagrama de flujo módulo LECTURA DE DATOS ANALÓGICOS

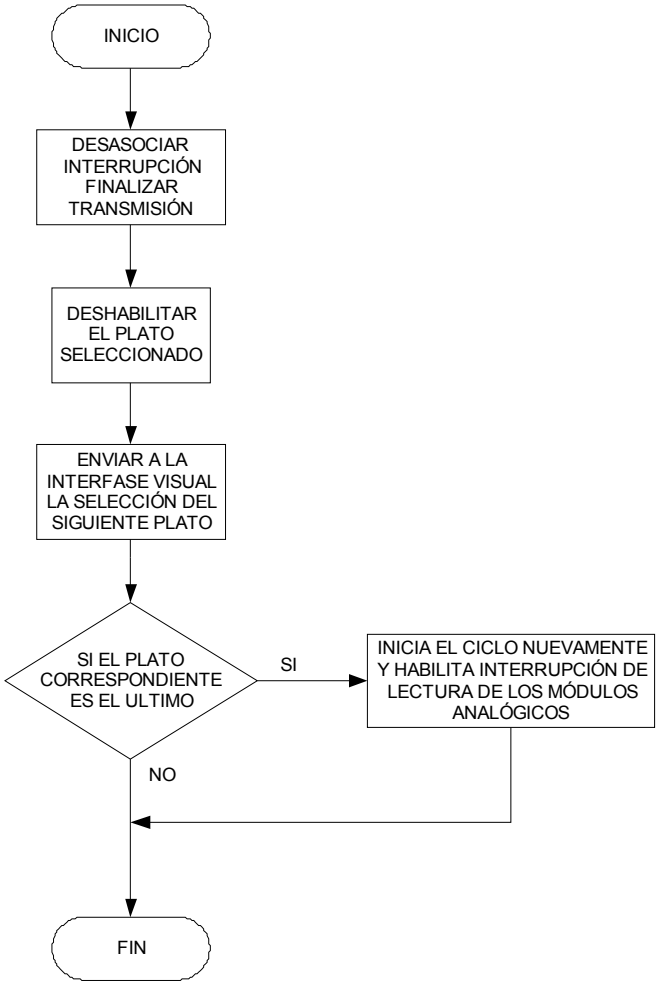
En la figura 25 se muestra el diseño de la lógica de algoritmo del módulo de lectura de datos analógicos, el código fuente en lenguaje de alto nivel:

Network 1 // Lectura de los datos en los módulos analógicos

```
LD    SM0.0
MOVW  AIW0, VW100
MOVW  AIW2, VW200
MOVW  AIW4, VW300
MOVW  AIW6, VW400
MOVW  AIW8, VW500
MOVW  AIW10, VW600
MOVW  AIW12, VW700
```

Módulo control de finalización de envío:

FIGURA 26: Diagrama de flujo módulo CONTROL DE FINALIZACION



En la figura 26 se muestra el diseño de la lógica de algoritmo del módulo de control de finalización de envió, el código fuente en lenguaje de alto nivel:

Network 1 // Desasociar interrupción Finalizar transmisión

```
LD     SM0.0
DTCH   9
```

Network 2 //Deshabilita el plato 1 si esta activo y envía la solicitud de envío de datos del plato siguiente.

```
LD     M3.1
XMT    VB2, 0
R      M3.1, 1
```

Network 3 // Deshabilita el plato 2 si esta activo y envía la solicitud de envío de datos del plato siguiente.

```
LD     M3.2
XMT    VB4, 0
R      M3.2, 1
```

Network 4 // Deshabilita el plato 3 si esta activo y envía la solicitud de envío de datos del plato siguiente.

```
LD     M3.3
XMT    VB6, 0
R      M3.3, 1
```

Network 5 // Deshabilita el plato 4 si esta activo y envía la solicitud de envío de datos del plato siguiente.

```
LD     M3.4
XMT    VB8, 0
R      M3.4, 1
```

Network 6 // Deshabilita el plato 5 si esta activo y envía la solicitud de envío de datos del plato siguiente.

```
LD     M3.5
XMT    VB10, 0
R      M3.5, 1
```

Network 7 // Deshabilita el plato 6 si esta activo y envía la solicitud de envío de datos del plato siguiente.

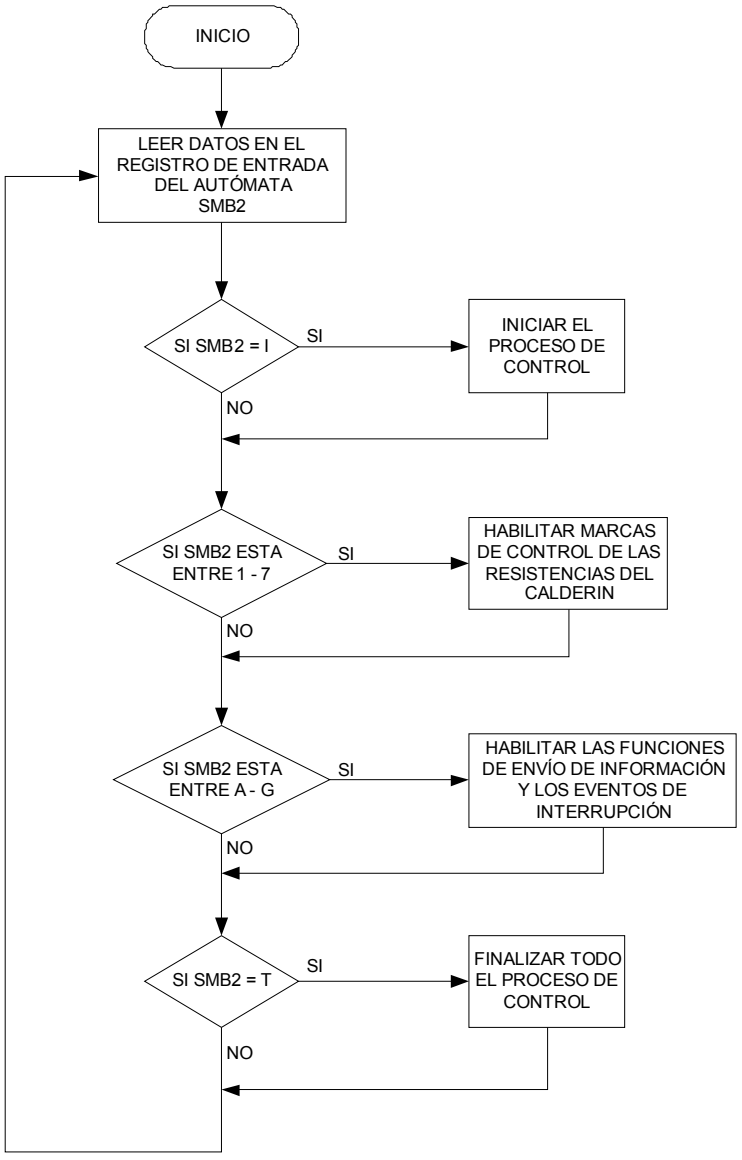
```
LD     M3.6
XMT    VB12, 0
R      M3.6, 1
```

Network 8 // Deshabilita el plato 7 si esta activo y habilita la lectura de los datos de los módulos analógicos al finalizar la transmisión a la interfase visual.

```
LD     M3.7
XMT    VB0, 0
ATCH   LECTURA, 10
R      M3.7, 1
```

Modulo de datos de entrada:

FIGURA 27: Diagrama de flujo del módulo DATOS DE ENTRADA



En la figura 27 se muestra el diseño de la lógica de algoritmo del módulo de datos de entrada, el código fuente en lenguaje de alto nivel:

Network 1 // Iniciar el proceso
LDB= SMB2, 'I'
CALL INICIALIZAR

Network 2 // Datos en el registro de entrada que habilitan la marca de control de R1
LDB= SMB2, '1'
S M1.1, 1

Network 3 // Datos en el registro de entrada que habilitan la marca de control de R2
LDB= SMB2, '2'
S M1.2, 1

Network 4 // Datos en el registro de entrada que habilitan la marca de control de R3
LDB= SMB2, '3'
S M1.3, 1

Network 5 // Datos en el registro de entrada que habilitan la marca de control de R4
LDB= SMB2, '4'
S M1.4, 1

Network 6 // Datos en el registro de entrada que habilitan la marca de control de R5
LDB= SMB2, '5'
S M1.5, 1

Network 7 // Datos en el registro de entrada que habilitan la marca de control de R6
LDB= SMB2, '6'
S M1.6, 1

Network 8 // Datos en el registro de entrada que habilitan la marca de control de R7
LDB= SMB2, '7'
S M1.7, 1

Network 9 // Datos en el registro de entrada que ejecuta las instrucciones en el modulo de finalizar el proceso de destilación
LDB= SMB2, 'F'
CALL FINALIZAR

Network 10 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'A'
S M3.1, 1
ATCH CTRLREAD, 9
DTCH 10
ATCH OUTDATA, 11

Network 11 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'B'
S M3.2, 1
ATCH CTRLREAD, 9
ATCH OUTDATA, 11

Network 12 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'C'
S M3.3, 1
ATCH CTRLREAD, 9
ATCH OUTDATA, 11

Network 13 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'D'
S M3.4, 1
ATCH CTRLREAD, 9
ATCH OUTDATA, 11

Network 14 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'E'
S M3.5, 1
ATCH CTRLREAD, 9
ATCH OUTDATA, 11

Network 15 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'F'
S M3.6, 1
ATCH CTRLREAD, 9
ATCH OUTDATA, 11

Network 16 // Datos en el registro de entrada que habilitan las funciones de envió de información y los eventos de interrupción
LDB= SMB2, 'G'
S M3.7, 1
ATCH CTRLREAD, 9
ATCH OUTDATA, 11

3.3. Hardware

El diseño se realizará utilizando:

- SIMATIC S7–200 CPU 224 con una alimentación nominal de AC 120 a 240 V y de 14 entradas DC 24 V y 10 salidas de relé y dos módulos de ampliación EM 231 Termopar, 4 entradas analógicas.

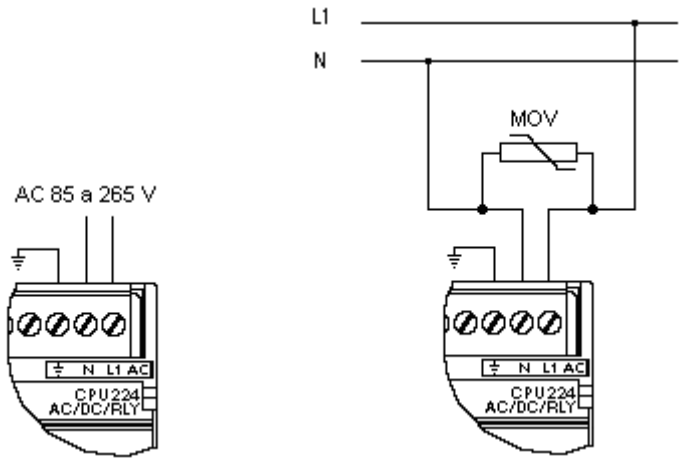
- Un ordenador tipo PC, con Windows XP, 2000, NT, 98 o 95 como sistema operativo, 64MB de memoria RAM como mínimo, puerto de comunicación serial RS-232.

- Cable PC/PPI que conecta el S7-200 con la unidad de programación como se puede observar en la figura 28.

FIGURA 28: Conexión del cable PC/PPI

La figura 29 muestra el cableado de una CPU S7-200 con alimentación AC, la utilización de un varistor de óxido metálico (MOV) limitará la tensión de pico y ofrecerá protección interna a los circuitos internos del sistema de automatización S7-200.

FIGURA 29: Alimentación del S7-200 CPU 224



La tensión de trabajo del varistor MOV será como mínimo de un 20% superior a la tensión nominal de fase.

3.3.1. Interfases de entrada

Los dispositivos que se utilizarán para obtener la información de temperatura en cada plato de la torre y el entendimiento del autómata y la computadora son los siguientes:

- 7 termopares de 2 hilos tipo J 2 blindados
- 2 módulos EM 231 Termopar de 4 entradas cada uno
- Cable PC/PPI

Estas interfaces trabajarán de la siguiente forma:

Los termopares se forman cuando se unen dos metales distintos que, al calentarse, generan una fuerza electromotriz. La tensión generada es proporcional a la temperatura de unión. Se trata de una tensión pequeña; un micro voltio puede representar varios grados. La base de la medición de temperatura utilizando termopares consiste en medir la tensión de un termopar, compensar las uniones adicionales y linealizar posteriormente el resultado.

El módulo sirve para conectar el S7-200 a señales analógicas de nivel bajo en un rango de ± 80 mV. Todos los termopares conectados al módulo deben ser del mismo tipo.

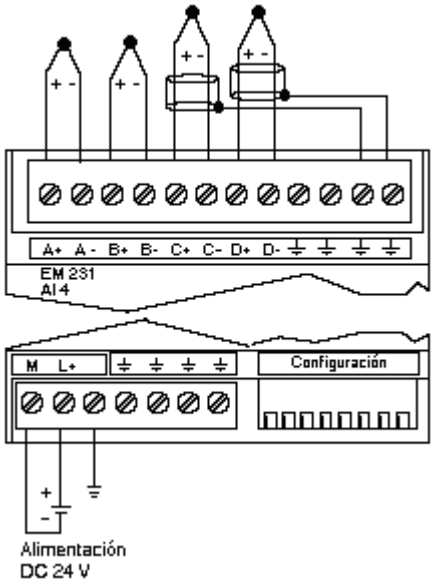
PPI es un protocolo maestro-esclavo. Los maestros envían peticiones a los esclavos y éstos responden. Los esclavos no inician mensajes, sino que esperan a que un maestro les envíe una petición o solicite una respuesta. Las redes S7-200 utilizan el estándar RS-485 con cables de par trenzado.

El cable PC/PPI y el modo Freeport se pueden utilizar para conectar las CPUs S7-200 a numerosos dispositivos compatibles con el estándar RS-232.

El cable PC/PPI se encuentra en modo de transmisión cuando los datos se envían del puerto RS-232 al RS-485. El cambio, se encuentra en modo de recepción al estar inactivo, o bien cuando los datos se transmiten del puerto RS-485 al RS-232. El cable cambia inmediatamente de modo de recepción a transmisión cuando detecta caracteres en el canal de transmisión del RS-232.

En la figura 30 se muestra la forma de conexión de cada hilo blindado y no blindado al módulo EM 231 Termopar

FIGURA 30: Identificación de terminales de conexión



La configuración de los interruptores DIP del módulo Termopar se muestra en la figura 31.

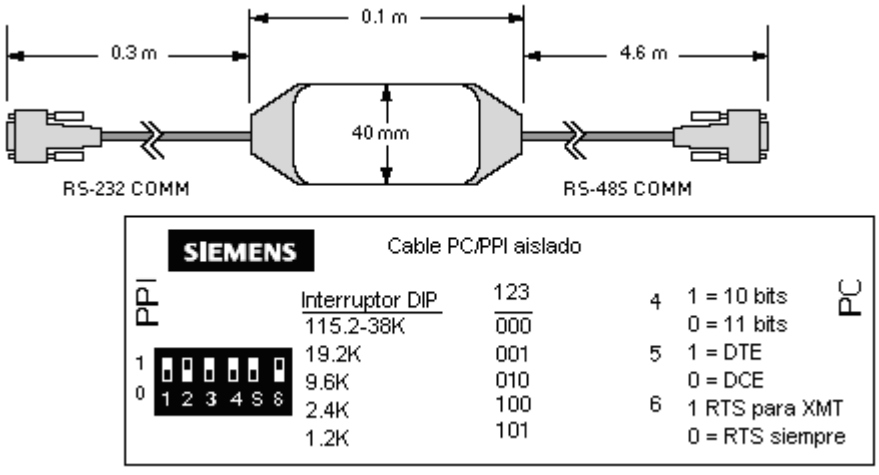
FIGURA 31: Interruptor DIP del módulo EM 231 Termopar

00001100

- Tipo de termopar J
- Detección de hilos abiertos
- Habilitar la detección de hilos abiertos
- Escala de temperatura grados Celsius
- Habilitar la compensación de la temperatura en la unión fría

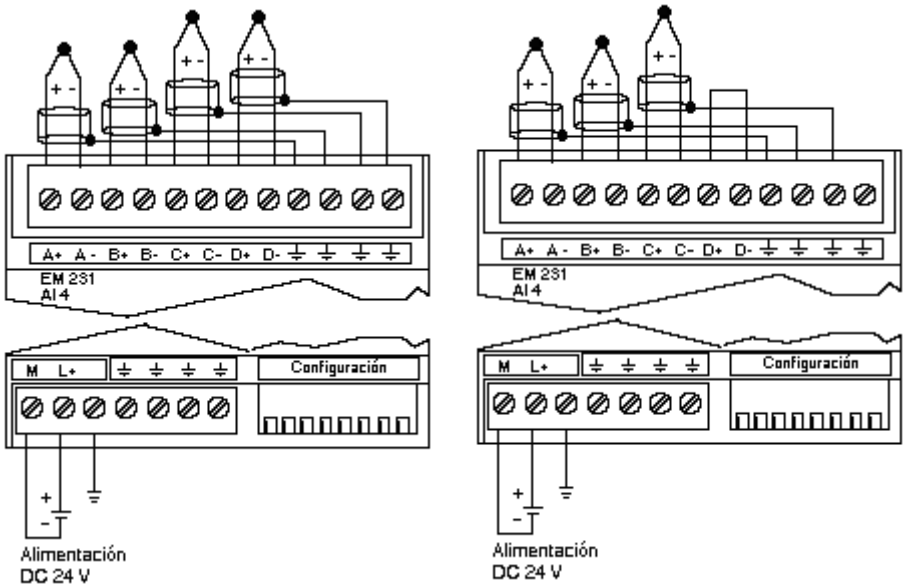
En la figura 32 se muestra las dimensiones del cable PC/PPI y la configuración del interruptor DIP en modo de 9.6K.

FIGURA 32: Cable PC/PPI



Se usarán dos módulos conectados como se muestra en la figura 33, usando hilos blindados que vienen de cada plato de la torre de un total de 7, sobrará un canal en el módulo 2; estas entradas serán cortocircuitadas.

FIGURA 33: Conexión final de los módulos EM 231 Termopar

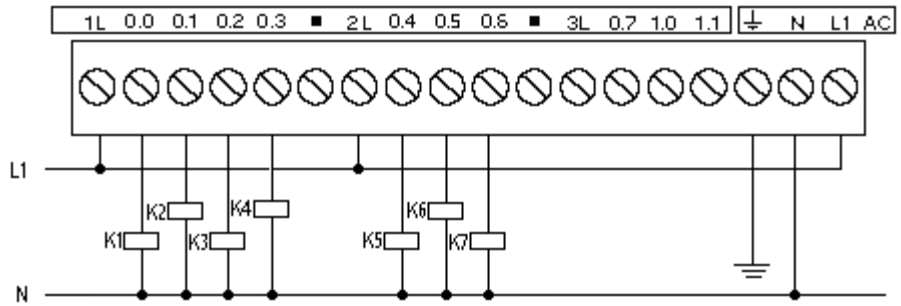


3.3.2. Interfase de salida

Se utilizarán 7 contactores sin contactos auxiliares para conmutar las resistencias en el calentador del calderín, la figura 34 muestra el cableado de salida del S7-200.

- K1 Resistencia 1 del calentador
- K2 Resistencia 2 del calentador
- K3 Resistencia 3 del calentador
- K4 Resistencia 4 del calentador
- K5 Resistencia 5 del calentador
- K6 Resistencia 6 del calentador
- K7 Resistencia 7 del calentador

FIGURA 34: Diagrama de cableado de la CPU 224



El cable PC/PPI también será usado como medio de comunicación exterior entre el S7-200 y la PC ver figura 32.

3.4. Implantación del sistema

Ahora ya están listas las condiciones para implantar el sistema; lo normal es que esto implique una "conversión" del sistema existente al nuevo. El enfoque de una organización a la conversión de sistema depende de que tanto se desee aceptar el riesgo y de la cantidad de tiempo disponible para la conversión.

3.4.1. Conversión en paralelo

En la conversión en paralelo, el sistema existente y el nuevo operan simultáneamente, o en paralelo, hasta que le equipo de proyecto tiene confianza de que el nuevo sistema esté trabajando de manera adecuada. Las otras dos ventajas clave de la conversión en paralelo son: (1) el sistema ya existente sirve de respaldo, en caso de que el nuevo sistema no opere en la forma esperada y (2) los resultados del nuevo sistema pueden compararse con los resultados del existente. La desventaja obvia de la conversión en paralelo es la doble carga de trabajo impuesta al personal y a los recursos de hardware.

3.4.2. Conversión directa

Conforme se mejoran los procedimientos de prueba del sistema, se empieza a tener mayor confianza en la capacidad para implantar un sistema que funcione, cambiando la conversión en paralelo por una conversión directa. En este caso existe un mayor riesgo asociado debido a que no hay respaldo si el sistema llega a fallar.

Cuando carece de un sistema existente o cuando el sistema actual es muy diferente se elige "en frío" este enfoque.

3.4.3. Conversión en fases

En la conversión en fases, se implanta un sistema, módulo por módulo, ya sea por conversión en paralelo o directa, tiene la ventaja de diseminar la demanda de los recursos, de manera que no sea tan pesada como resultará en un momento dado. Las desventajas son que: (1) la conversión tarda más, y (2) debe existir una interfaz de sistema entre el existente y el nuevo después de que se implanta cada nuevo módulo.

3.4.4. Conversión piloto

En la conversión piloto, el nuevo sistema se implanta por conversión en paralelo, directa o en fases como sistema "piloto" sólo en una de las varias áreas para las que esté destinado. Por ejemplo, supóngase que en una compañía se desea implantar un sistema de planeación de recursos de manufactura en sus ochos plantas. Podría elegirse una planta como piloto y primero implantarla ahí el nuevo sistema.

La ventaja de la conversión piloto es que los inevitables errores ocultos de un sistema pueden eliminarse antes de implantarlo en otros lugares.

Las desventajas del proyecto de conversión piloto es que el tiempo de implantación del sistema total tardará más que si todo el sistema se implantara de una sola vez.

4. PRÁCTICA DE DESTILACIÓN DE UNA SOLUCIÓN BINARIA

La operación unitaria de destilación es un método que se usa para separar los componentes de una solución líquida, que depende de la distribución de estos componentes entre una fase vapor y una fase líquida. Ambos componentes están presentes en ambas fases. La fase vapor se origina de la fase líquida por vaporización al punto de ebullición.

El requerimiento básico para separación de los componentes por destilación, consiste en que la composición del vapor sea diferente de la composición del líquido con el cual está en equilibrio al punto de ebullición de este último. La destilación se basa en soluciones en las que todos los componentes son bastante volátiles, como soluciones amoníaco-agua o etanol-agua, donde ambos componentes estarán también, en la fase vapor.

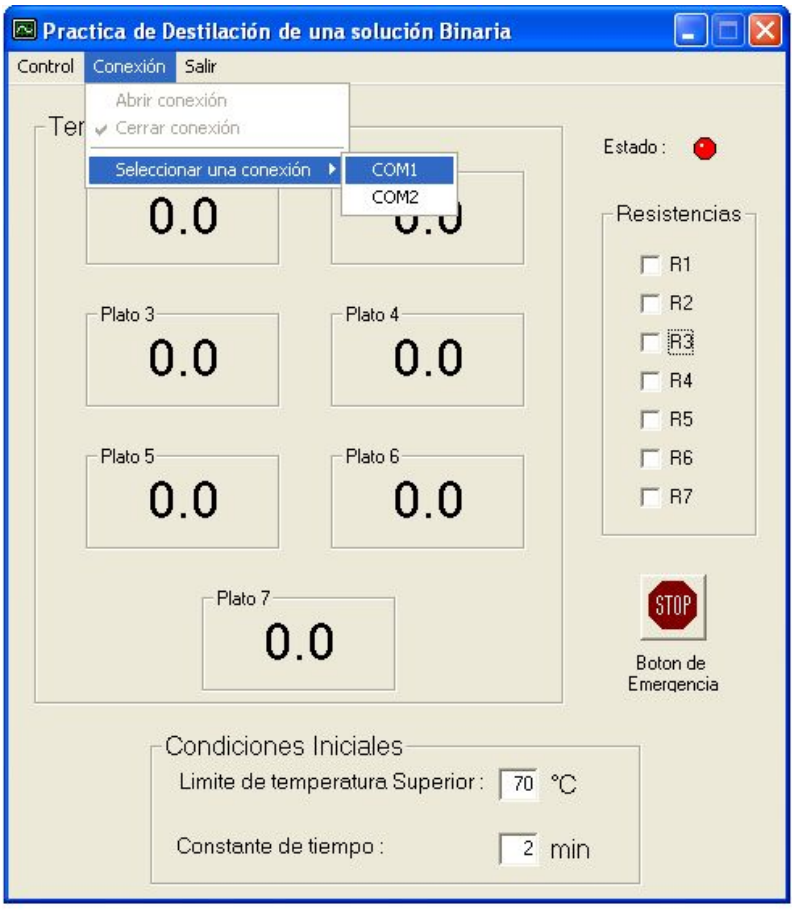
La destilación puede llevarse a cabo en la práctica por medio de cualquiera de dos métodos principales. El primer método consiste en la producción de un vapor por ebullición de la mezcla líquida que se va a separa en una sola etapa, recuperando y condesando los vapores. El segundo método implica el retorno de una porción del condesado al destilador. Los vapores se desprenden a través de una serie de etapas o platos a contracorriente con respecto a los vapores. A este segundo método se le llama destilación fraccionada.

4.1. Inicio

Para iniciar la práctica es necesario que los programas (interfase visual y programa interno lógico) estén activos. Después de que la solución a destilar este en el calderín de la destiladora, el estudiante sólo tendrá contacto con la interfase visual durante todo el proceso de destilación.

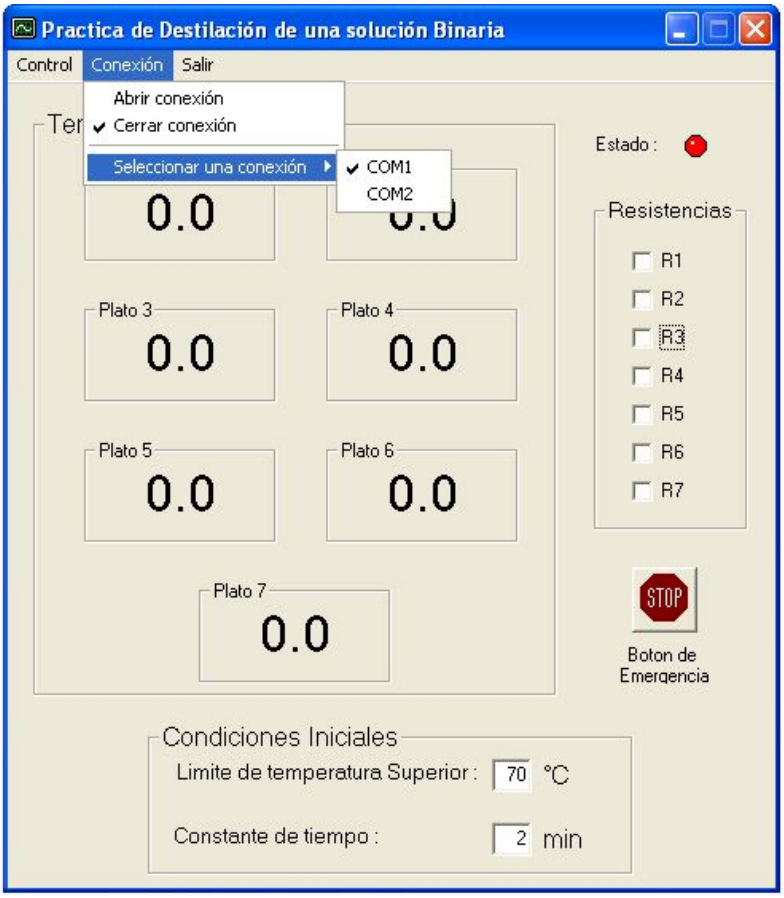
Elegir primeramente el puerto de comunicación a usar como se puede observar en la figura 35.

FIGURA 35: Seleccionar puerto de comunicación



Se habilitan las opciones de abrir conexión y cerrar conexión mostrando el estado de la conexión elegida como puede observarse en la figura 36

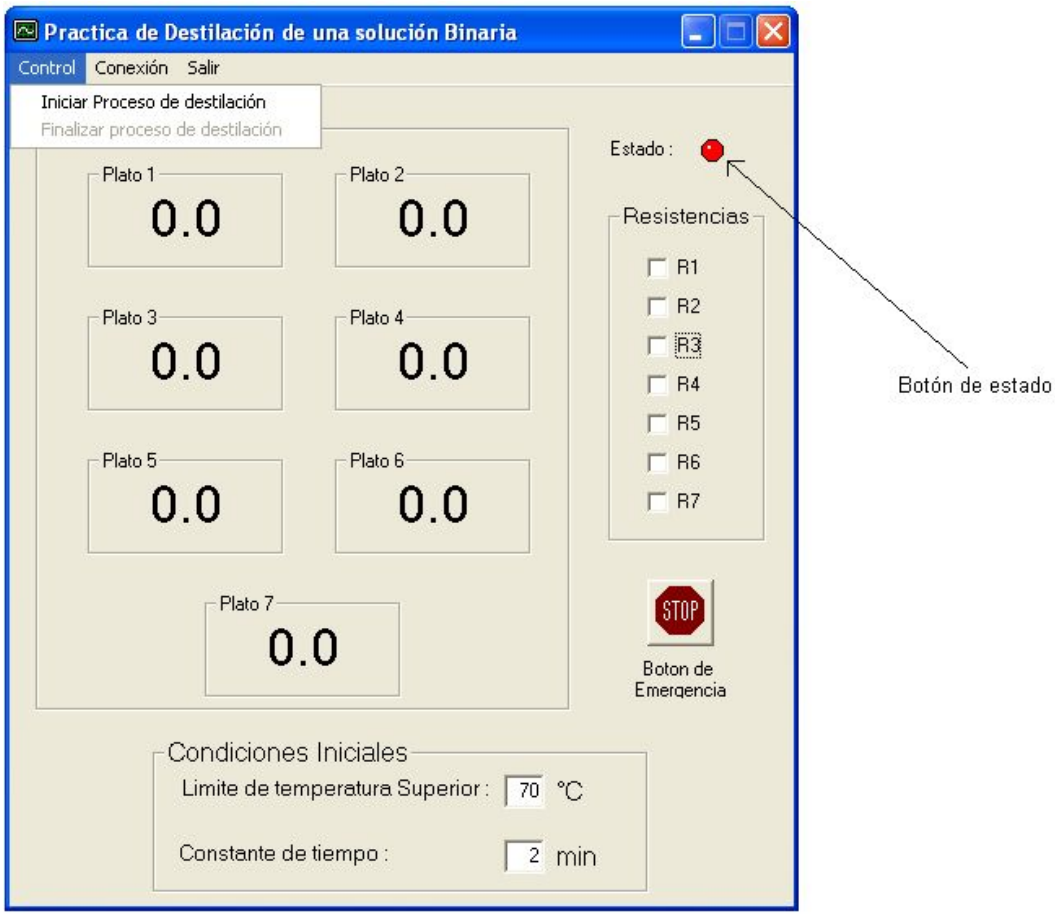
FIGURA 36: Verificar conexión elegida.



Abrir la conexión previamente antes de iniciar el proceso de destilación, para que el programa habilite las opciones de iniciar proceso de destilación y el botón de estado.

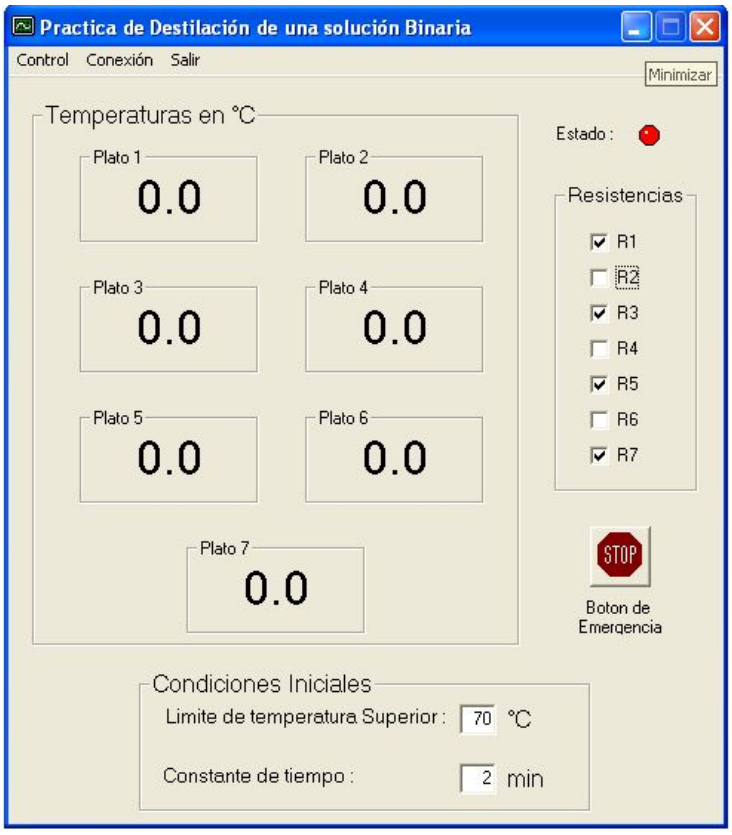
El proceso puede iniciar de dos formas: por medio del menú de control o por un clic del Mouse en el indicador de estado como se puede observar el figura 37.

FIGURA 37: Inicio de proceso de destilación



Previo a iniciar el proceso, se debe elegir la combinación de las resistencias a usar en la práctica como también el límite de temperatura superior y la constante de tiempo. Por default, el programa muestra en estas casillas de ingreso 70 grados Celsius para el límite de temperatura superior y 2 minutos como constante de tiempo, como puede observarse en la figura 38.

FIGURA 38: Opciones de ingreso

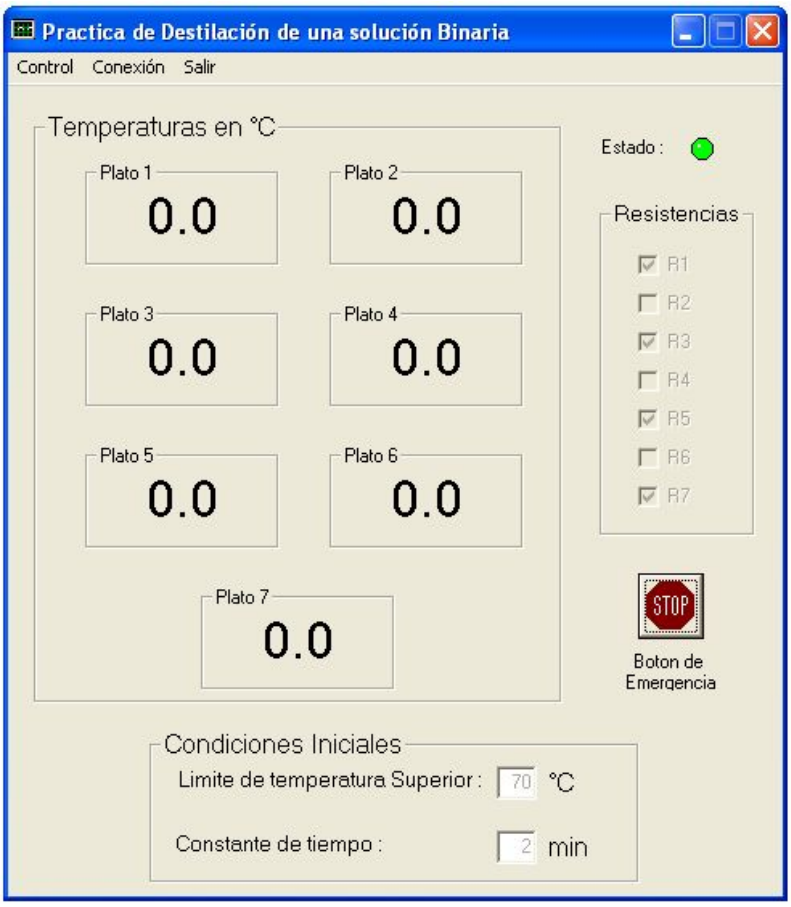


Ahora se inicia el proceso de destilación por las dos diferentes opciones de inicio mencionadas anteriormente. Ver figura 37.

4.2. Desarrollo

Cuando el programa inicia el proceso de destilación, se deshabilitan parcialmente las opciones de ingreso de inicio de la práctica, como se puede observar en la figura 39.

FIGURA 39: Opciones parcialmente deshabilitadas.



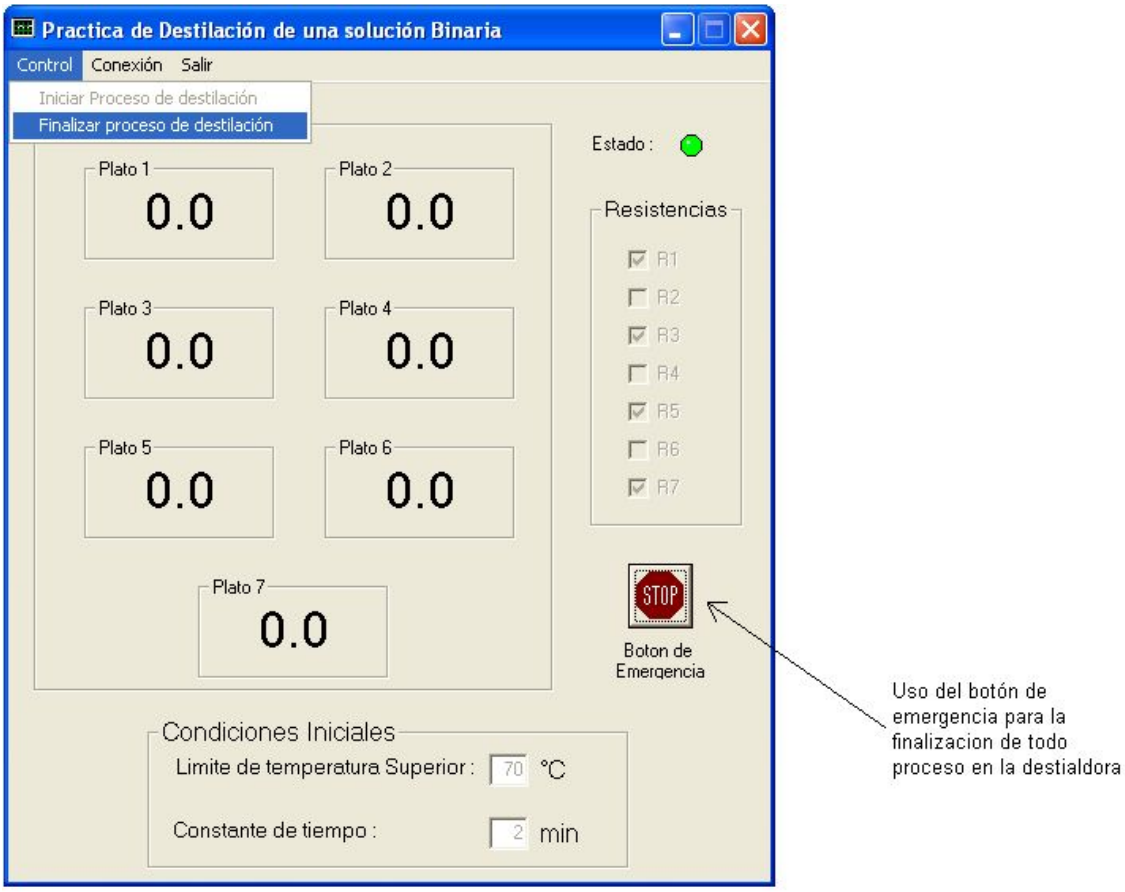
El progreso de la práctica se realizará por si sola, es decir, los que están realizando el trabajo de destilación no tendrán que estar presentes durante todo el proceso; el sistema realizará todas las tareas necesarias para llevar a cabo el proceso de destilación.

En el desarrollo de la práctica, el programa (interfase visual) mostrará en lugares específicos las temperaturas existentes en cada plato de la torre en intervalos de 1 minuto.

4.3. Finalización

El sistema automáticamente finalizará los procesos iniciados en la destiladora cuando se logren las condiciones de equilibrio térmico en cada plato de la torre, pero si es necesario es posible finalizarlo manualmente, como puede observarse en la figura 40.

FIGURA 40: Finalización de la práctica



CONCLUSIONES

1 Un algoritmo puede ser utilizado para facilitar el inicio del diseño de hardware y software necesarios para la implementación de un proyecto industrial automatizado.

2 Para la automatización de un sistema industrial es elemental aplicar técnicas de modularización y programación apropiadas que posibiliten su correcta ejecución y mantenimiento.

3 El uso del principio de refinamiento descendente: Top down, en el diseño de proyectos automatizados proporciona una solución adecuada y funcional para proyectos extensos.

4 El diseño estructurado es un enfoque disciplinado para diseñar software y hardware más claros y fáciles de probar, depurar y modificar que los proyectos no estructurados.

5 Es primordial depurar y probar cada subprograma en proyectos extensos cuando, finalmente, estén terminados y no hasta que el proyecto haya sido codificado por completo.

6 En el diseño de programas, la correcta aplicación de las técnicas de programación lograrán que un programa sea fácil de leer y de comprender, logrando que el mantenimiento y modificaciones sean apropiados.

7 La integración de la práctica de destilación de una solución binaria en forma automatizada y estructurada proporcionará a los futuros profesionales reducir, significativamente, la inspección humana, el cansancio y el margen de error, aumentando, así, la seguridad para el personal que la realiza.

RECOMENDACIONES

1 Al momento de diseñar sistemas automatizados, se deben considerar los requerimientos del usuario para evitar diseños incompletos.

2 Cuando un diseño se inicia, debe procurarse administrar el tiempo en el desarrollo del proyecto de tal forma que no se evada el uso de algoritmos y diagramas de flujo.

3 El principio de refinamiento descendente: Top down, es un método de diseño que todo diseñador debe de utilizar en sus proyectos, para evitar dificultades en la ejecución y mantenimiento del mismo.

4 Cuando se está codificando cada subprograma, se debe considerar la documentación en el momento de la escritura para evitar perdidas de tiempo tratando de recordar la lógica de diseño.

5 Enfatizar en los cursos del área de ingeniería electrónica y eléctrica la utilización de métodos estructurados que permitan al estudiante realizar proyectos de software y hardware sin dificultad.

BIBLIOGRAFÍA

AWL y KOP para SIMATIC S7-200 programación de bloques. s.e. s.l. Siemens, 1996. 430pp.

Geankoplis, Chistie. **Procesos de transporte y operaciones unitarias.** 2ª ed. México: CECSA, 1995. 830pp.

Kruse, Robert L. **Estructura de datos y diseño de programas.** México: Prentice Hall, 1988. 488pp.

Long, Larry. **Introducción a la informática y al procesamiento de información.** México: Prentice Hall, 1987. 566pp.

Tenenbaum, Aaron M. y Augenstein, Moshe. **Estructura de datos en pascal.** México: Prentice Hall, 1983. 560pp.

SIMATIC Manual del sistema de automatización S7-200. 4ª ed. s.l. Siemens, 2004. 568pp.

Tremblay, Jean Paul y Bunt, Richard. **Introducción a la ciencia de las computadoras enfoque algorítmico**. México: McGraw Hill, 1988. 486pp.