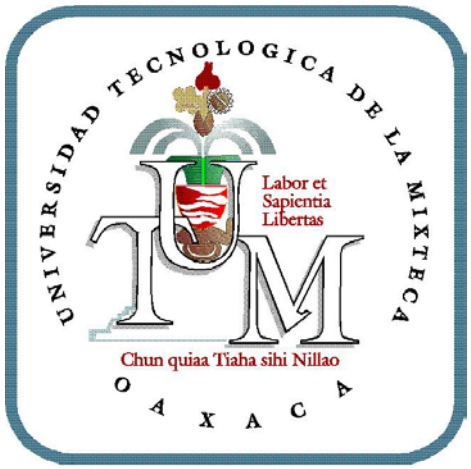
UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

INSTITUTO DE ELECTRÓNICA Y COMPUTACIÓN



Elementos de la Programación Estructurada: Algoritmos, Pseudo Código y Diagramas de Flujo

Ricardo Ruiz Rodríguez

Huajuapan de León Oaxaca
Septiembre 2003

Contenido

Índice de Ilustraciones

Índice de Tablas

Elementos de la Programación Estructurada: Algoritmos, Pseudo código y Diagramas de Flujo

1 Introducción.

Aunque este documento tiene como finalidad principal la presentación y desarrollo de tres temas: algoritmos, pseudo código y diagramas de flujo, la presentación de los mismos de manera asilada no se consideró como la mejor opción por lo que, si bien no se desarrollarán de manera exhaustiva, se presentarán conceptos relacionados con el paradigma de la programación estructurada, con la finalidad de proporcionar un marco de referencia propicio para los conceptos a desarrollar.

Se comenzará definiendo algunos conceptos básicos dentro del contexto de la programación estructurada, para después presentar y desarrollar los temas que constituyen la parte central de este trabajo y que ya fueron mencionados con anterioridad.

2 Programación Estructurada: panorama general.

La resolución de un problema mediante una computadora, se refiere al proceso consistente en partir de la descripción de un problema (habitualmente en lenguaje natural, y expresado en términos propios del dominio del problema), y desarrollar un programa de computadora que resuelva dicho problema. Este proceso exige, grosso modo, los siguientes pasos o etapas:

1. Comprensión del problema.
2. Análisis del problema.
3. Diseño o desarrollo de un algoritmo.
4. Transformación del algoritmo en un programa (codificación).
5. Ejecución y validación del programa.

Cada uno de estos pasos es de significativa importancia, sin embargo, la modesta experiencia en la enseñanza del paradigma de la programación estructurada indica, que es en las etapas 2 y 3 en donde el estudiante tiene mayor problema, por lo que estas etapas son con toda seguridad, dos de las más importantes en este proceso.

El orden en el que están puestas estas etapas no es aleatorio, cada una de las etapas supone la adecuada terminación de la anterior, así por ejemplo, sería imposible tratar de analizar un problema que ni siquiera se ha comprendido en su contexto más general, ni que decir entonces de la elaboración de un algoritmo.

Probablemente la etapa más sencilla de todas sea la número cuatro, ya que una vez que se ha comprendido, analizado el problema y obtenido un algoritmo que lo resuelve, su transformación a un programa de computadora es una tarea de simple traducción, que si bien requiere el conocimiento de la sintaxis del lenguaje elegido para ello, la parte intelectual y de raciocinio queda expresada en el algoritmo.

Según Jeffrey Ullman [Ullman76], **la meta de la programación estructurada** es la de hacer programas que sean fáciles de diseñar, depurar y susceptibles de ser entendidos por personas diferentes a las que escribieron el programa. No se trata de escribir programas que solamente su creador entienda, se trata de algo más que eso.

Ahora bien, un programa puede ser visto como un sistema en el que sus componentes son programas más pequeños, y a su vez estos componentes pueden ser vistos como otros, compuestos por similares todavía más pequeños y así sucesivamente hasta un nivel razonable, en donde los componentes finales tienden a consistir de unas cuantas sentencias. Los programas escritos en esta forma se denominan **programas estructurados** [Ullman76].

En un programa estructurado el flujo lógico de ejecución se gobierna por las estructuras de control básicas: **secuenciales, selectivas y repetitivas**. Estas estructuras constituyen una de las partes medulares de la programación estructurada, por lo que se mencionan con más detalle en las secciones siguientes.

2.1 Teorema de Böhm y Jacopini.

El teorema de Böhm y Jacopini dice que un programa propio puede ser escrito utilizando únicamente tres tipos de estructuras de control: estructuras secuenciales, estructuras de selección y estructuras de repetición.

Para que la programación sea estructurada, los programas han de ser propios. Un programa se define como **propio** si cumple las siguientes condiciones:

- Tiene un solo punto de entrada y un solo punto de salida.
- Todas las sentencias del algoritmo son alcanzables, esto es, existe al menos un camino que va desde el inicio hasta el fin del algoritmo.
- No posee ciclos infinitos.

De este teorema se deduce que, si los algoritmos se diseñan empleando exclusivamente dichas estructuras de control, los algoritmos y por consecuencia los programas derivados de ellos, serán propios.

Finalmente, antes de comenzar con una descripción más detallada de los algoritmos, se describirán muy brevemente estas tres estructuras de control.

2.2 Estructuras de control.

Las estructuras secuenciales, selectivas y repetitivas se denominan estructuras de control porque son las que controlan el modo o flujo de ejecución del algoritmo. Su importancia es tal, que una vez que se entienda su estructura y funcionamiento, puede decirse que en esencia es todo lo que hay que saber respecto al control y flujo de los algoritmos. Otro aspecto importante consiste en saber en dónde utilizarlas, pero esto es muy dependiente del problema a resolver, pero la buena noticia es que el mismo nombre de las estructuras identifica su objetivo.

2.2.1 Estructuras secuenciales.

Las estructuras secuenciales son las más simples de las tres; se caracterizan porque una acción o sentencia se ejecuta detrás de otra, esto es, el flujo del algoritmo coincide con el orden físico en el que se han puesto las sentencias del algoritmo.

2.2.2 Estructuras de selección.

Como su mismo nombre lo indica, este tipo de estructuras realizan una selección de las acciones o sentencias a ejecutar, es decir, dependiendo de si se cumple o no una determinada condición, se ejecutan o no, un determinado grupo de sentencias.

Es importante mencionar que la condición que rige el control del fujo puede ser tan elaborada como la naturaleza del problema lo requiera, pero se deberá tener la garantía de que el valor final de la condición podrá ser evaluado como un valor *booleano*, esto es,

como una condición **verdadera** o **falsa**, ya que de lo contrario la condición tendrá ambigüedad y, como se verá más adelante, esto viola una de las características más importantes de un algoritmo (véase la página 7).

Las estructuras de selección pueden ser simples, dobles o múltiples. Los flujos que se siguen dependiendo de la evaluación de la condición, pueden visualizarse más fácilmente al observar sus representaciones en diagrama de flujo (página 11) o en pseudo código (página 14), pero estos detalles han sido pospuestos para secciones posteriores.

2.2.3 Estructuras de repetición.

En las estructuras de repetición, los enunciados del cuerpo del ciclo se repiten mientras se cumpla una determinada condición, misma que deberá seguir los mismos lineamientos descritos para la estructura de selección descrita con anterioridad.

En este tipo de estructuras, es frecuente el uso de contadores[1] o centinelas[2] para controlar el número de repeticiones de un ciclo.

Más adelante en el texto, se muestran las notaciones algorítmicas utilizadas para las estructuras de repetición (páginas 12 y 15).

3 Algoritmos.

3.1 Concepto de algoritmo.

El concepto de algoritmo[3] forma parte esencial de los fundamentos de la computación. La matemática discreta y en particular la matemática constructivista es tan antigua como la propia matemática, esta última trata aquellos aspectos para los cuales existe una solución constructivista, esto es, no se conforma con demostrar la existencia de solución, sino que se pregunta cómo encontrar dicha solución [Abellanas91].

1 Utilizados cuando se conoce de antemano el número de repeticiones que el ciclo ejecutará.

2 Utilizados cuando no se sabe con certeza cuantas repeticiones habrá antes de que el ciclo termine.

3 El término algoritmo es muy anterior a la era de la computación: proviene de *Mohammed al-Khowârizmî* (cuyo apellido se tradujo al latín en la palabra *algoritmus*), quien era un matemático persa del siglo IX que enunció paso a paso las reglas para sumar, restar, multiplicar y dividir números decimales. En este mismo contexto también debe recordarse el algoritmo de Euclides obtenido 300 a.C.

En términos generales puede definirse un algoritmo como el método para resolver un determinado problema. El ejecutor de las instrucciones que realizan la tarea correspondiente se llama **procesador**. Existen algoritmos que describen toda clase de procesos, por ejemplo, las recetas de cocina, las partituras musicales, etc. Un procesador realiza un proceso siguiendo o ejecutando el algoritmo correspondiente. La ejecución de un algoritmo requiere la ejecución de cada uno de los pasos o instrucciones que lo constituyen.

Un algoritmo debe estar expresado de tal forma que el procesador lo entienda para poder ejecutarlo. Se dice que el procesador es capaz de interpretar el algoritmo, si el procesador puede realizar lo siguiente:

1. Entender lo que significada cada paso.
2. Llevar a cabo la sentencia correspondiente.

Esto significa que para que un algoritmo pueda ser correctamente ejecutado, cada uno de sus pasos debe estar expresado de tal forma que el procesador sea capaz de entenderlos y ejecutarlos adecuadamente.

Ahora bien, el término de algoritmo puede tener diferentes connotaciones dependiendo del contexto en el que se hable, por lo que es importante mencionar que el contexto en el que se aplicará en este texto es en el de la programación, esto es, el de utilizar a la computadora como herramienta para la resolución de problemas.

En el contexto que nos compete, se definirá un **algoritmo** como un conjunto finito de instrucciones que especifican la secuencia ordenada de operaciones a realizar para resolver un problema.

En base a lo anterior, si el procesador del algoritmo es una computadora, el algoritmo debe estar expresado en forma de un **programa**, el cual se escribe en un **lenguaje de programación[4].** A la actividad de expresar un algoritmo en un lenguaje de programación determinado se le denomina **programar**.

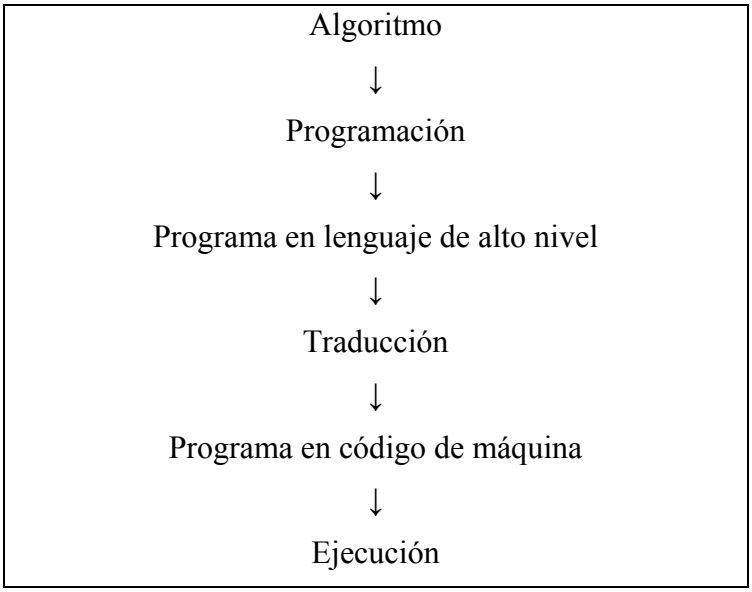
4 Un lenguaje de programación es lenguaje artificial que se utiliza para expresar programas de computadora.

Cada paso del algoritmo se expresa mediante una instrucción o sentencia en el programa, por lo que un programa consiste en una secuencia de instrucciones en donde cada una de las cuales especifica ciertas operaciones a realizar por la computadora.

3.2 Uso de la computadora en la resolución de problemas.

En general, se escriben algoritmos para resolver problemas que no son tan fáciles de resolver a primera vista y de los que necesitamos especificar el conjunto de acciones que se llevarán a cabo para su resolución. Además, como lo que interesa es resolver problemas utilizando la computadora, los algoritmos tendrán como finalidad ser traducidos en programas, por lo que es conveniente mencionar el proceso general de resolución de problemas desde que se dispone de un algoritmo hasta que la computadora lo ejecuta. El proceso mostrado en la Tabla 1, ha sido adaptado de [Abellanas91].

Tabla 1 Proceso general de resolución de problemas con la computadora.



Existen diferentes formas de traducir un algoritmo a un programa. En el diagrama mostrado en la Tabla 1, se ha escogido la representación en un lenguaje de alto nivel porque los lenguajes de este tipo proporcionan un mayor nivel de abstracción y familiarización con el lenguaje natural (inglés). Aquí también es importante recordar que las computadoras tienen su propio lenguaje (binario), por lo que es necesario un proceso de traducción (realizado por el compilador) para que se traduzca el conjunto de sentencias escritas en un lenguaje de programación de alto nivel (código fuente) a un conjunto de

instrucciones que sean compresibles para la computadora (código de máquina). Finalmente, y aunque el proceso involucrado es más complejo que lo que aquí se ha descrito, si todo esta bien, el resultado de este proceso será la ejecución del programa basado en el algoritmo.

Teniendo en consideración lo anterior, debería ser claro que el papel del algoritmo es fundamental, ya que sin algoritmo no puede haber programas y sin programas no hay nada que ejecutar en la computadora.

Es importante mencionar y enfatizar también, que un algoritmo es independiente tanto del lenguaje de programación en que se exprese, como de la computadora en la que se ejecute, por lo que no deberían ser diseñados en función de ello.

Otro aspecto muy importante dentro de los algoritmos, es el del análisis y diseño de algoritmos. El diseño formal de algoritmos es una actividad intelectual difícil y complicada en general, ya que el diseño de buenos algoritmos requiere creatividad e ingenio, y no existen reglas para diseñar algoritmos en general, esto es, todavía no existe un algoritmo para diseñar algoritmos y que además éstos sean buenos, por lo que, aunque es importante mencionarlo y tenerlo en cuenta, no se cubrirá en este texto la parte relacionada al análisis y diseño de algoritmos.

3.3 Cinco importantes condiciones de un algoritmo.

Los algoritmos, además de ser un conjunto finito de reglas que dan lugar a una secuencia de operaciones para resolver un tipo específico de problemas, deben cumplir con cinco importantes condiciones [Abellanas91] mismas que son descritas en la Tabla 2.

Las cinco condiciones mostradas en la Tabla 2, reducen significativamente el espectro tan amplio que hasta ahora se ha manejado como algoritmo. Así por ejemplo, aunque una receta de cocina podría considerarse como un algoritmo, es común encontrar expresiones imprecisas en ella que dan lugar a ambigüedad (violando la condición 2), tales como “añádase una pizca de sal”, “batir suavemente”, etc., invalidando con ello la formalidad de un algoritmo dentro del contexto que nos interesa.

Tabla 2 Cinco condiciones con las que debe cumplir un algoritmo.

1. **Finitud**:	Un algoritmo tiene que acabar siempre tras un número finito de pasos. (Un procedimiento que tiene todas las características de un algoritmo salvo que posiblemente falla en su finitud, se conoce como método de cálculo.)
2. **Definibilidad**:	Cada paso de un algoritmo debe definirse de modo preciso; las acciones a realizar han de estar especificadas para cada caso rigurosamente y sin ambigüedad.
3. **Conjunto de entradas**:	Debe existir un conjunto especificado de objetos, cada uno de los cuales constituye los datos iniciales de un caso particular del problema que resuelve el algoritmo. A este conjunto se le denomina conjunto de entradas del algoritmo.
4. **Conjunto de salidas**:	Debe existir un número especificado de objetos, cada uno de los cuales constituye la salida o respuesta que debe obtener el algoritmo para los diferentes casos particulares del problema. A este conjunto se le denomina conjunto de salidas del algoritmo. Para cada entrada del algoritmo, debe existir una salida asociada que constituye la solución al problema particular determinado por dicha entrada.
5. **Efectividad**:	Un algoritmo debe ser efectivo. Esto significa que todas las operaciones a realizar por el algoritmo deben ser lo bastante básicas para poder ser efectuadas de modo exacto, en un lapso de tiempo finito por el procesador que ejecute el algoritmo.

3.4 Estructura de un algoritmo.

Aunque no existe una única forma de representar un algoritmo[5], la estructura general de un algoritmo debería ser como la mostrada en la Tabla 3.

5 Véase [Abellanas91], [Deitel95], [Joyanes93], [Ullman76], [Wirth86], etc. cada uno tiene formas diferentes aunque esencialmente iguales de representar un algoritmo

Tabla 3 Estructura general de un algoritmo.

Algoritmo <*nombre_del_algoritmo*>
Inicio
definición de constantes
declaración de variables
Sentencia 1
Sentencia 1
.
.
.
Sentencia n
Fin

Aunado a la estructura general del algoritmo propuesta en la Tabla 3, se considera recomendable que, partiendo de la descripción del problema y de un análisis inicial, se determine la entrada, la salida y el proceso general del algoritmo.

A continuación se menciona un ejemplo clásico (el algoritmo de Euclides) con la finalidad de ilustrar el proceso descrito.

Definición del problema.

Dados dos números enteros positivos *m* y *n*, encontrar su **máximo común divisor**, es decir, el mayor entero positivo que divide a la vez a *m* y a *n*.

Entrada: Dos números enteros positivos: *m* y *n*.

Salida: Un número que representa el MCD (Máximo Común Divisor) de *m* y *n*.

Proceso: La solución está basada en el residuo de la división de los operandos *m* y *n*. Si el residuo es 0 entonces hemos terminado, si no, habrá que hacer intercambio de valores y continuar con el proceso.

El ejemplo anterior ilustra el proceso de resolución que se debería seguir para resolver un problema, ya que partiendo de este análisis inicial, la idea será entonces ir particularizando cada parte en un refinamiento progresivo, hasta llegar a un algoritmo más

refinado y funcional. En las secciones posteriores se continuará trabajando con este ejemplo modelando el algoritmo en su notación de diagrama de flujo y de pseudo código.

3.5 Pruebas de algoritmos.

Una vez que se ha generado un algoritmo que parece correcto, una de las partes más importantes dentro de su diseño es la referente a las pruebas. La parte de la validación de los datos de entrada al algoritmo es también un aspecto importante, aunque normalmente lo que se hace es construir un algoritmo aparte que se encargue de validar que los datos de entrada sean los correctos, pero ese aspecto no es lo que se quiere comentar en esta sección.

Una vez que se tiene una solución algorítmica de un problema, no se debería suponer o creer que funcionará bien siempre. En el diseño del algoritmo se deben considerar al menos algunos casos de prueba. Es habitual que el domino de trabajo de un algoritmo sea un conjunto de elementos y entonces sería bueno saber por ejemplo ¿Cómo se comporta el algoritmo en los límites del conjunto? ¿Dado un mismo dato de entrada obtengo siempre la salida esperada? entre otras preguntas.

La fase de prueba de los algoritmos es una parte fundamental dentro del diseño del mismo, y adoptarlo como práctica se recomienda como una importante técnica de programación y como esencial principio de Ingeniería de Software.

Las técnicas y métodos formales de pruebas están fuera de los alcances de este trabajo introductorio al paradigma de la programación estructurada, pero a este nivel, basta con saber que es conveniente realizar algunas pruebas sobre los algoritmo desarrollados. Un libro ampliamente recomendable en cuanto a técnicas básicas de pruebas y estilo de programación es [Kernighan00].

4 Diagramas de Flujo.

Un diagrama de flujo es un tipo de notación gráfica algorítmica.

Un **diagrama de flujo** es una herramienta gráfica de descripción de algoritmos que se caracteriza por utilizar un conjunto de símbolos gráficos y expresar de forma clara los flujos de control u orden lógico en el que se realizan las acciones de un algoritmo.

Aunque existe en la literatura una amplia variedad de representaciones para los símbolos utilizados en los diagramas de flujo, en este texto se adoptaran sólo cinco, mismos que se presentan en la Ilustración 1.

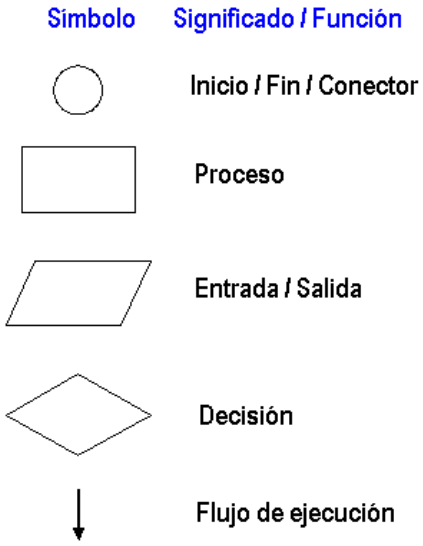


Ilustración 1 Símbolos y su significado en los diagramas de flujo.

4.1 Estructuras de control.

Esta sección muestra los diagramas de flujo de las estructuras de control, para más detalles al respecto refiérase a la página 3.

4.1.1 Estructura secuencial.

La Ilustración 2 muestra el diagrama de flujo que representa a la estructura de control secuencial.

Ilustración 2 Estructura secuencial en diagrama de flujo.

4.1.2 Estructuras de selección.

La Ilustración 3 muestra los diagramas de flujo de las estructuras de selección.

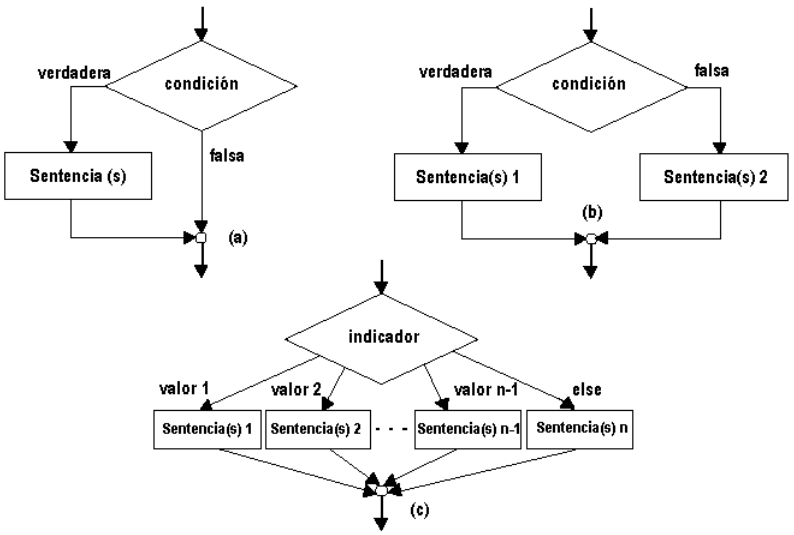


Ilustración 3 Estructuras de selección en diagrama de flujo a) simple, b) doble y c) múltiple.

Puede observarse en la Ilustración 3 que en la estructura de selección simple se evalúa la condición, y si ésta es verdadera, se ejecuta un determinado grupo de sentencias; en caso contrario, las sentencias son ignoradas. En la estructura de selección doble, cuando la condición es verdadera, se ejecutará un determinado grupo de sentencias, y si es falsa se procesará otro grupo diferente de sentencias. Por último, en la estructura de selección múltiple se ejecutarán unas sentencias u otras según sea el valor que se obtenga al evaluar una expresión representada por el indicador. Se considera que dicho resultado ha de ser de tipo ordinal, es decir de un tipo de datos en el que cada uno de los elementos que constituyen el tipo, excepto el primero y el último, tiene un único predecesor y un único sucesor.

4.1.3 Estructuras de repetición.

La Ilustración 4 muestra las estructuras de repetición básicas.

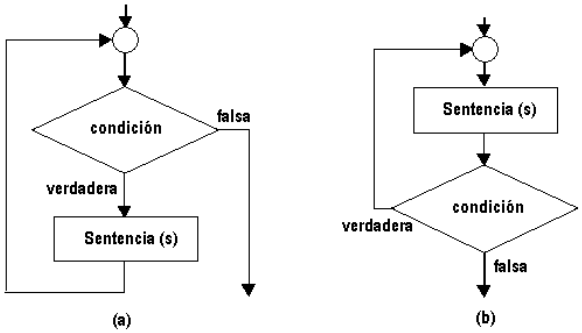


Ilustración 4 Estructuras de repetición en diagrama de flujo a) hacer mientras (while) y b) repetir hasta (do-while).

Lo que caracteriza a la estructura de repetición "hacer mientras", es que los enunciados del cuerpo del ciclo se realizan cuando la condición es verdadera, además de que se pregunta por la condición al principio, de donde se deduce que las sentencias se podrán ejecutar de 0 a N veces. En la estructura de repetición "repetir hasta", las sentencias del interior del ciclo se ejecutan una vez y continúan repitiéndose hasta que la condición sea falsa. La verificación de la condición se realiza al final del ciclo, por lo que se deduce que las sentencias se ejecutarán al menos una vez y hasta un máximo de N.

4.2 Diagrama de flujo del algoritmo de Euclides.

La Ilustración 5 muestra el diagrama de flujo para el problema propuesto en la sección 3.4. La solución al problema está determinada por el algoritmo de Euclides.

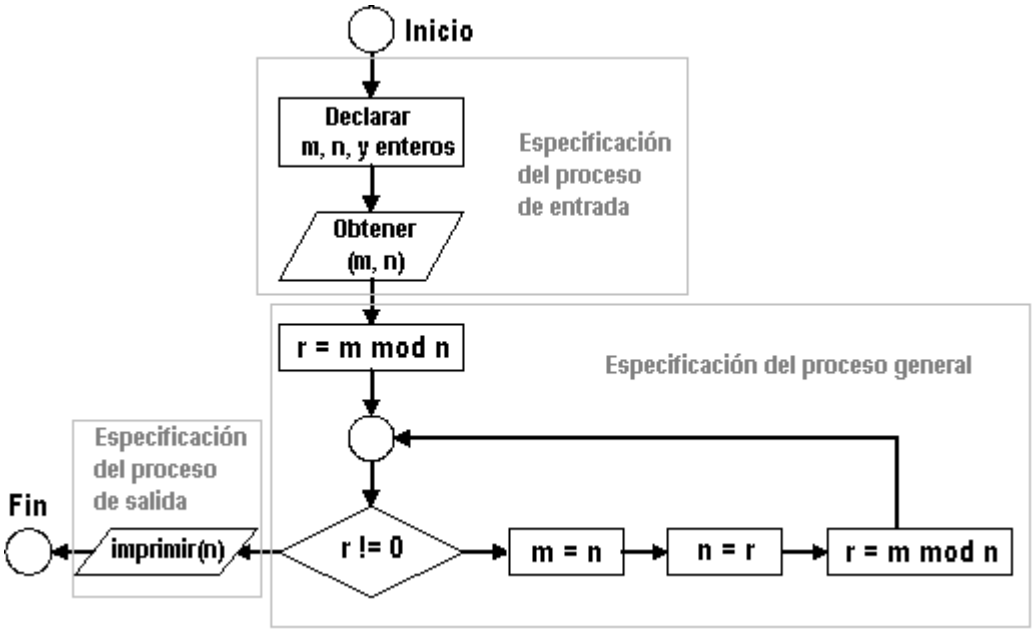


Ilustración 5 Diagrama de flujo para el algoritmo de Euclides.

Para ser congruentes con la propuesta de solución realizada en la sección 3.4, además del algoritmo se ha indicado en recuadros de menor intensidad las especificaciones del proceso de entrada, del proceso de salida y del proceso general.

5 Pseudo código

El pseudo código es otro tipo de notación algorítmica textual.

El pseudo código es un lenguaje artificial de especificación de algoritmos caracterizado por:

- Mantener una identación o sangría adecuada para la fácil identificación de los elementos que lo componen.
- Permitir la declaración de los datos (constantes y/o variables) manipulados por el algoritmo.
- Disponer de un conjunto pequeño y completo de palabras reservadas que permitan expresar: las acciones elementales, las primitivas de composición de acciones y la definición de acciones con nombre.

El pseudo código se concibió para superar las dos principales desventajas del diagrama de flujo: lento de crear y difícil de modificar sin un nuevo diagrama. Es una herramienta muy buena para el seguimiento de la lógica de un algoritmo, y para transformar con facilidad los algoritmos a programas escritos en un lenguaje de programación específico.

5.1 Estructuras de control.

A continuación se muestran las principales estructuras selectivas y de repetición en su representación en pseudo código. Todas las estructuras siguen las mismas características descritas con anterioridad (véanse las páginas 3 y 11), por lo que en esta sección la descripción de las mismas sólo se limita a su presentación.

5.1.1 Estructura secuencial.

La Tabla 4 ilustra la representación en pseudo código de la estructura secuencial.

Tabla 4 Estructura secuencial en pseudo código.

<Sentencia 1>
<Sentencia 2>
.
.
.
<Sentencia n>

5.1.2 Estructuras de selección.

Las estructuras de selección simple, doble y compuesta se muestran respectivamente en la Tabla 5, Tabla 6 y la Tabla 7.

Tabla 5 Estructura de selección simple en pseudo código.

```
if <condición> then
        <Sentencia(s)>
```

Tabla 6 Estructura de selección doble en pseudo código.

```
if <condición> then
        <Sentencia(s) 1>
else
        <Sentencia(s) 2>
```

Tabla 7 Estructura de selección múltiple en pseudo código.

```
if <condición 1> then
        <Sentencia(s) 1>
else if <condición 2> then
        <Sentencia(s) 2>
                .
                .
                .
else
        <Sentencia(s) n>
```

```
case <indicador> of
        <valor 1>   : <Sentencia(s) 1>
        <valor 2>   : <Sentencia(s) 2>
                        .
                        .
                        .
        <valor n-1> : <Sentencia(s) n-1>
        <else>      : <Sentencia(s) n>
end case
```

5.1.3 Estructuras de repetición.

Las estructuras de repeticón “hacer mientras” y “repetir hasta” son presentadas en la Tabla 8 y la Tabla 9 respectivamente.

Tabla 8. Estructura de repetición hacer mientras (while).

```
while <condición> do
        <Sentencia(s)>
end while
```

Tabla 9 Estructura de repetición repetir hasta (do-while).

```
do
        <Sentencia(s)>
while <condición>
```

5.2 Pseudo código del algoritmo de Euclides.

Finalmente, esta sección presenta el pseudo código para el problema propuesto en la sección 3.4. Se recomienda comparar el algoritmo propuesto en la Tabla 10 con el diagrama de flujo de la Ilustración 5.

Tabla 10 Algoritmo de Euclides en pseudo código.

```
Algoritmo MCD
Inicio
        variables
                m, n, r de tipo entero

        obtener (m, n)
        r = m mod n
        while (r ≠ 0) do
                m = n
                n = r
                r = m mod n
        end while
        imprimir (n)
Fin
```

6 Diseño básico de programas estructurados.

La esencia del análisis mostrado a continuación, está basado principalmente en las reglas para la formación de programas estructurados propuestas en el libro de Deitel [Deitel95]. Así mismo, es importante también hacer mención de que, debido a la mejor visualización proporcionada por los diagramas de flujo gracias a sus representaciones gráficas, la presentación de esta sección se basa principalmente en ellos, pero los conceptos también son aplicables al pseudo código.

El conectar de forma arbitraria y/o poco cuidadosa símbolos individuales de diagramas de flujo, puede incurrir en diagramas de flujo no estructurados (véase la Ilustración 6) y por consiguiente en programas no estructurados. La idea entonces es,

utilizar estructuras de control de una entrada y una salida[6] de tal manera que sólo haya una forma de entrar y una de salir de la estructura de control.

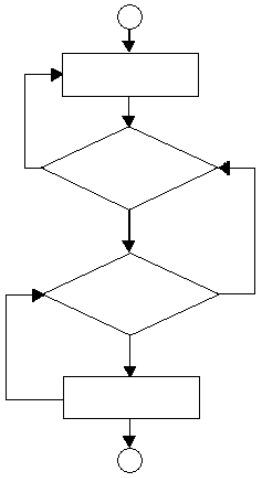


Ilustración 6 Diagrama de flujo NO estructurado.

6.1 Reglas para la formación de algoritmos estructurados.

En la Tabla 11 se muestran las reglas para la construcción de algoritmos estructurados (adaptadas de [Deitel95]). Estas reglas bien podrían denominarse "*Algoritmo para la construcción de algoritmos estructurados*".

Tabla 11 Reglas para la formación de algoritmos estructurados.

1. Empiece con el diagrama de flujo más simple.
2. Cualquier rectángulo (acción, sentencia, etc.) puede ser reemplazado por dos rectángulos de manera secuencial. Esta es la regla de apilamiento.
3. Cualquier rectángulo puede ser reemplazado por cualquier estructura de control. Esta es la regla de anidamiento.
4. Aplicar de manera sucesiva las reglas 2 y 3

6 Las estructuras mostradas en las secciones anteriores cumplen con esto y antes de continuar se debería estar convencido de ello. Obsérvese que, aunque la estructura se selección múltiple mostrada en la Ilustración 3 c) parece tener más de una salida, sólo es procesada una de ellas y que, independientemente de la sentencia(s) que haya(n) sido seleccionada(s), su salida converge a un punto de salida común representado por el conector.

La Ilustración 7 muestra la regla 1 y la aplicación repetida de la regla 2. Obsérvese que la regla 1 es perfectamente consistente con el proceso general de resolución de algoritmos propuesto en la sección 3.4.

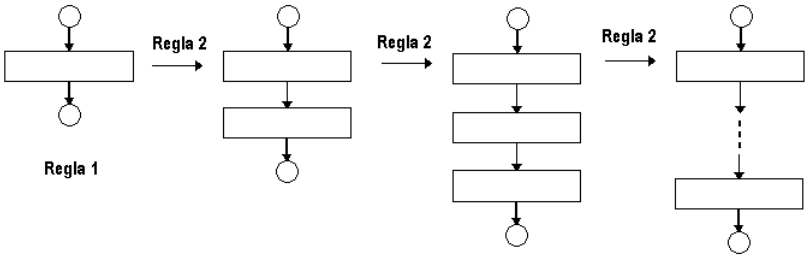


Ilustración 7 Aplicación de las reglas 1 y 2.

La aplicación de las reglas de la Tabla 11 siempre resulta en un diagrama de flujo estructurado con una apariencia nítida y de bloques constructivos [Deitel95]. La Ilustración 8 muestra la aplicación de la regla de anidamiento al más simple diagrama de flujo. Los bloques han sido substituidos por estructuras de selección doble y de repetición.

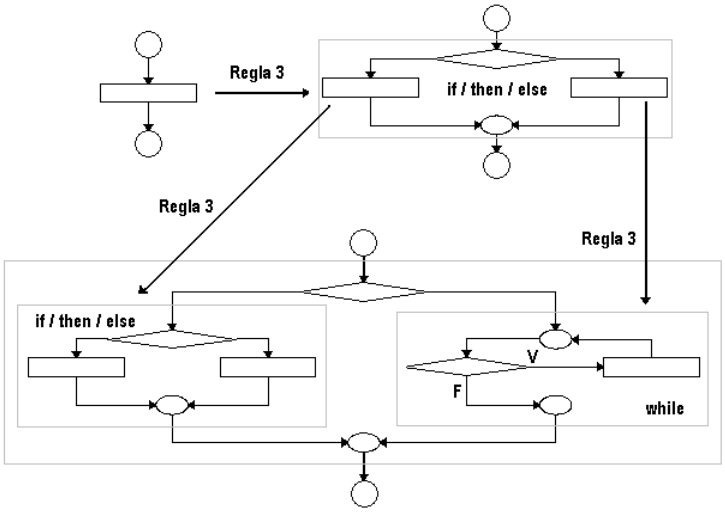


Ilustración 8 Aplicación de la regla 3.

Por último, y ya para finalizar con la presentación de los temas, es importante resaltar que la regla 4 genera estructuras más grandes, más complejas y más profundamente anidadas. Además, los diagramas de flujo que resultan de aplicar las reglas de Tabla 11 constituyen el conjunto de todos los diagramas de flujo estructurados y, por lo tanto, también el conjunto de todos los posibles programas estructurados [Deitel95].

7 Conclusiones.

La programación estructurada es un paradigma que no sólo vale la pena conocer, sino que es imprescindible saberlo como parte de la formación integral dentro del campo de la computación no sólo por su enfoque, sino porque sigue siendo ampliamente utilizado por su misma naturaleza[7], dentro de diversas áreas y disciplinas como la programación de sistemas, sistemas de tiempo real, etc.

Por otro lado, no debe confundirse a la programación estructurada con el conocimiento de un lenguaje de programación, la programación estructurada es un modelo de programación, y es independiente del lenguaje que se utilice para su implementación.

Finalmente, el estudio y conocimiento de técnicas y algoritmos previos a la fase de implementación, se recomienda como una buena práctica de programación y como principio esencial de la Ingeniería de Software, esto es, antes de enfrentarse en una lucha encarnizada contra la computadora con el lenguaje de programación como arma, se debería, antes de pasar a la implementación, hacer un análisis concienzudo del problema a resolver, elaborar un plan de solución general, especificar gradualmente dicho plan, y realizar un conjunto de pruebas representativas, si lo que se quiere es llevar a buen fin sus proyectos de programación, y evitar dolores de cabeza innecesarios.

7 La ejecución de programas implementados dentro del paradigma de la programación estructurada, hace que los programas sean más veloces que aquellos realizados dentro del paradigma de la programación orientada a objetos por ejemplo, esto debido a las características inherentes de los paradigmas.

8 Bibliografía.

[Abellanas91] Abellanas, M. y Lodares D. *Análisis de Algoritmos y Teoría de Grafos*. Macrobit ra-ma, 1991.

[Deitel95] Deitel, H. M. y Deitel, P. J. *Cómo programar en C / C++*. Prentice Hall, 1995.

[Joyanes93] Joyanes, Aguilar L. *Metodología de la Programación: Diagramas de flujo, Algoritmos y Programación Estructurada*. Mc Graw Hill, 1993.

[Kernighan00] Kernighan, B.W. y Pike R. *La práctica de la Programación*. Prentice Hall 2000

[Nist03] Nist. *Dictionary of Algorithms and Data Structures*. http://www.nist.gov/dads/ Septiembre 2003.

[Ullman76] Ullman, Jeffrey. D. *Fundamental Concepts of Programming Systems*. Addison-Wesley Publishing Company Inc, 1976.

[Wirth86] Wirth, Niklaus. *Algoritmos y Estructuras de Datos*. Prentice Hall, 1986.