ÍNDICE

2 Conceptos de Bases

2.1 Modelo de Von Neumman

Cada una de las computadoras que fueron creándose a lo largo de cada una de las generaciones de las computadoras (incluyendo las actuales) presenta un modelo en particular, es decir, dentro de cada una de las computadoras se tienen catalogados los dispositivos en elementos de entrada, salida, memoria y una unida de procesamiento (procesador). A este modelo se le conoce como el **modelo de Von Neumman**, y esta constituido por 4 elementos que están parcialmente interconectados entre ellos.

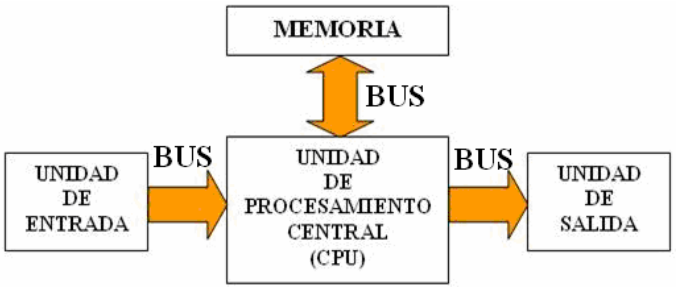


Figura 2.- Modelo de Von Neumann.

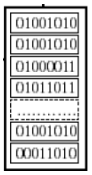
Este modelo representa la arquitectura de cualquier computadora, aún las actuales. Cada uno de los elementos del modelo representa características los cuales se describen a continuación:

- **Unidad de Entrada**
  Los dispositivos de entrada y salida (E/S) permiten la comunicación entre la computadora y el usuario [C]. Los dispositivos de entrada sirven para leer la información y ser almacenados temporalmente en la memoria y posteriormente la computadora efectué algunas operaciones con esos datos.
  Dispositivos de entrada: teclado, mouse, lector digita, escaner, cable de red, etc.

- **Unidad de Salida**
  Una vez que la computadora haya efectuado algunas operaciones los resultados obtenidos representan la salida, los cuales serán escritos en dispositivos de salida.
  Dispositivos de salida: impresora, monitor, cable de red, bocinas, etc.

- **Memoria**
  La memoria permite almacenar información que posteriormente será utilizada. El tipo de información que es posible almacenar son: ***Datos*** que pueden ser manipulados para efectuar alguna operación, ***Instrucciones*** que serán ejecutados por la unidad de procesamiento central y ***Direcciones*** de memoria que hacen referencias a instrucciones o datos dentro de la misma memoria.

  La memoria puede ser presentada como una tira que contiene valores '0' y '1' que representan cadenas binarias. Cada una de esas cadenas binarias representa instrucciones, datos o direcciones de memoria.

  Dispositivos de Memoria:
  ROM (Read Only Memory): CD, DVD, BIOS, etc.
  RAM (Random Access Memory): Discos Duros, USB, Memoria RAM, Cache, etc.

- **Unidad de Procesamiento Central (CPU)**

  Es considerado de gran importancia ya que representa el "cerebro de la computadora" que se encarga de efectuar una serie de operaciones referentes al calculo (suma, restas, divisiones, ciclos, selecciones, etc.). La CPU utiliza una serie de elementos o dispositivos para desarrollar sus funciones, estos se muestran en la siguiente figura.

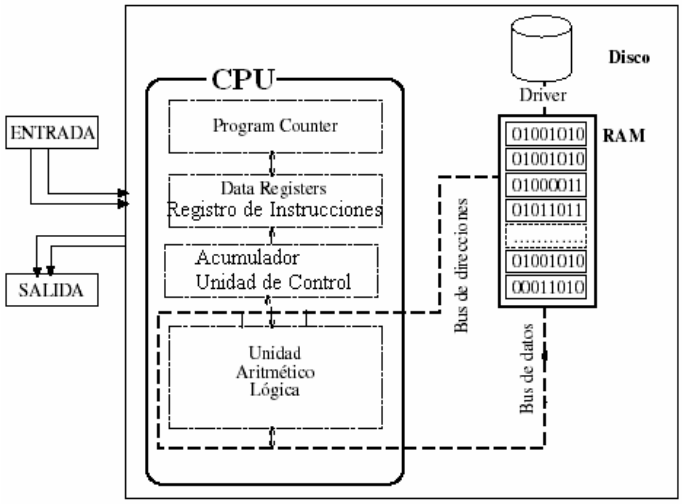


Figura 3.- CPU del Diagrama de Von Neumann.

- *Unidad Aritmética Lógica (ALU)*: Se encarga de efectuar operaciones aritméticas y lógicas básicas.
- *Acumulador (AC)*: Registro de almacenamiento que permite efectuar operaciones directamente con la ALU.
- *Contador de Programa (PC)*: Determina la posición en la memoria para obtener la instrucción, dato o dirección de memoria.
- *Registro de instrucciones (IR)*: Determina la posición en la memoria para la siguiente instrucción.
- *Unidad de Control (UC)*: Controla cada uno de los dispositivos dentro del CPU, así como los registros de entrada y salida que se conectan con los correspondientes dispositivos. Determina el orden, control y sincronización del proceso de la computadora.

- **Canal de Comunicación - BUS**

  Un bus es un conjunto de alambres paralelos usados para conectar los componentes de una computadora [10]. El bus permite el canal de comunicación entre los distintos componentes de una computadora, por un bus es posible que retransmitan una cierta cantidad de bits al mismo tiempo ya que esta constituido por una cantidad de pequeños canales paralelos, existen bus de 8, 16, 32 y 64 bits, estos últimos son los más actuales. Existen distintos buses: bus de datos, bus de control y bus de direcciones cada uno un funcionamiento especifico.

2.2 Algoritmo

Los programas están constituidos por una serie de instrucciones que efectúan un **proceso** en donde se requiere de **elementos de entrada** para efectuar el proceso y genera **elementos de salida** que representan los resultados del proceso.

A este proceso es conocido como algoritmo, un **algoritmo** es un método para resolver un problema que generalmente consiste en una serie de pasos.

Los algoritmos presentan las siguientes **características** [5]:

- Un algoritmo debe de ser **preciso** e indicar el orden (**secuencia**) de realización de cada paso.
- Un algoritmo debe de estar **definido o repetible**. Al desarrollar (ejecutar) el algoritmo dos veces, se debe de obtener el mismo resultado.
- Un algoritmo debe de ser **finito**. Debe de contener un número finito de pasos.

A continuación se describen algunos ejemplos de algoritmos.

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"
Entrada: Temperatura en grados Celsius (C).
Salida: Temperatura en grados Fahrenheit (F).
Procedimiento:

1.- Determinar el valor de la temperatura en Celsius C.
2.- Desarrollar la formula 9/5 * C + 32 y asignar el resultado a F.
3.- Indicar el resultado de la conversión contenido en F.

Ejemplo "**Promedio de 5 calificaciones**"
Entrada: 5 calificaciones {C1, C2, C3, C4, C5}.
Salida: Promedio de las calificaciones (P).
Procedimiento:

1.- Determinar el valor de cada una de las calificaciones.
2.- Desarrollar la formula (C1 + C2 + C3 + C4 + C5) / 5 y asignar el resultado a P.
3.- Indicar el resultado del promedio de calificaciones P.

Ejemplo "**Receta de Cocina**"
Objetivo : Preparar hot-cakes
Entrada: 1 huevo, 1 taza de leche, 1 cucharada de mantequilla, azúcar y sartén.
Salida: 10 hot-cakes.
Procedimiento:

1.- Mezclar los ingredientes.
2.- Batir mezcla hasta que no haya grumos.
3.- Dejar reposar 20 min.
4.-Colocar el sartén a fuego lento.
5.-Untar mantequilla sobre el sartén.
6.-Poner pasta y extender.
7.-Voltear hot-cake cuando sea pertinente.
8.-Retirar cuando este cocido por ambos lados.
9.-Repetir desde el paso 5, hasta terminar la pasta.

2.3 Software Base

2.3.1 Enfoque Usuario-Programador – (Sistemas Operativos)

Para comprender el funcionamiento de la computadora es necesario especificar los actores que se desenvuelven entorno a la utilización de la computadora, en la figura 1 (lado izquierdo) se representa a la computadora y un actor que interactúa con la computadora como un "**Usuario**".

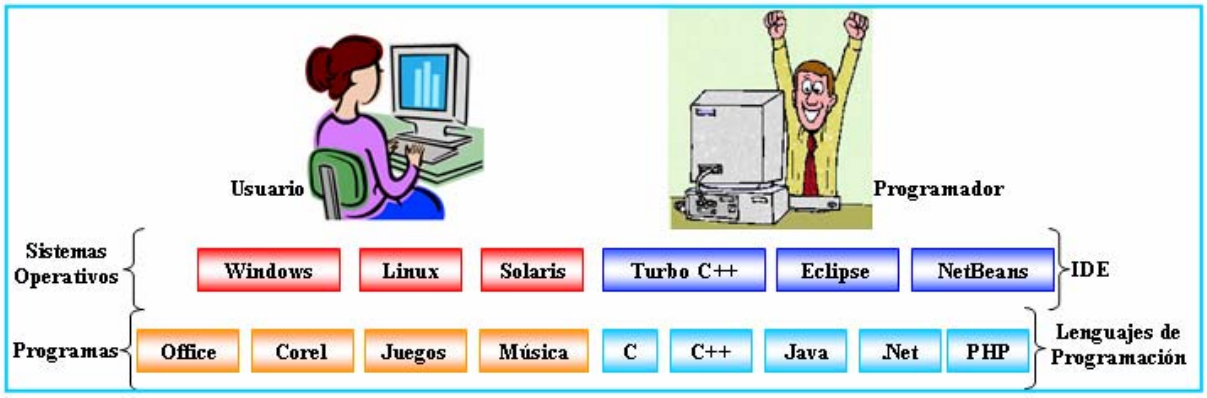


Figura 4.- Enfoque "Usuario –Programador".

El usuario interactúa con la computadora mediante el uso de programas. Un **programa** es una secuencia de instrucciones que describe cómo ejecutar cierta tarea [10], algunos ejemplos de programas son Word, Excel, Mathemathica, Corel, juegos, herramientas de simulación, exploradores, etc. Los programas se ejecutan sobre un **Sistema Operativo** el cual tiene como función administrar y organizar los recursos que se disponen en la computadora para mejorar su utilización [1], Los sistemas operativos dirigen las operaciones globales de las computadoras como pueden ser introducir y ejecutar nuevos programas. Algunos ejemplos de sistemas son Windows, MAC-OS, Linux, Unix, Solaris, etc.

Del lado derecho de la figura se encuentra otro actor que interactúa con la computadora pero efectuando otras actividades como escribir y diseñar los programas, a este actor se le conoce como "**Programador**". El programador hace uso de **algoritmos** (sección 2.2) que permiten describir paso a paso lo que el programa debe de ejecutar para resolver un problema en particular, dicho algoritmo debe de ser traducido a un programa mediante el uso de un lenguaje de programación y las operaciones que expresan el algoritmo en forma de programar se le conoce como **programación** [5].

2.3.2 Lenguajes de Programación

Los lenguajes utilizados para escribir programas de computadoras son los **lenguajes de programación** y los programadores son los escritores y diseñadores de programas. Dichos lenguajes de programación son escritos en un **programa o código fuentes** que presentan las siguientes características:

Análisis lexicográfico: utiliza expresiones regulares para determinar la estructura entre los caracteres o símbolos.

Análisis semántico: determina el significado de la estructura del código fuente, efectúa el reconocimiento de símbolos por medio de un diccionario específico al lenguaje de programación

Análisis sintáctico: determina la estructura gramatical del código fuente con el uso de tokens (conjunto de caracteres con un significado de acuerdo al lenguaje de programación.

Un **lenguaje de programación** es un lenguaje que son utilizados para escribir y diseñar programas. Existen tres niveles de lenguajes de programación (figura 6):

-Lenguaje máquina.
-Lenguaje de bajo nivel (ensamblador).
-Lenguajes de alto nivel.

La computadora hace uso de instrucciones proporcionadas por el usuario o el programador, sin embargo la computadora es un componente electrónico que hace uso de voltajes para la representación de la información. Esta representación hace referencia a un lenguaje binario en donde un voltaje alto representa el valor '1' y un voltaje bajo el valor'0'.

El **lenguaje máquina** es un lenguaje que hace uso de cadenas binarias (cadenas constituidas por símbolos o caracteres - '0' y '1'), para representar instrucciones o los datos involucrados en las operaciones. A las operaciones se les conoce como instrucciones de máquina o ***código máquina***. Este tipo de lenguaje es dependiente de la máquina en la que se esta programado impidiendo la portabilidad del programa [5].

Los **lenguajes de bajo nivel** dependen de la máquina en la que se programa, pero son más fáciles de utilizar a diferencia del lenguaje máquina. El lenguaje de bajo nivel más utilizado es el ***lenguaje ensamblador***, el cual hace uso de neumónicos para las instrucciones por ejemplo: ADD, SUB, DIV (suma, resta, división).

Los **lenguajes de alto nivel** son utilizados por la mayoría de los programadores ya que presentan un diseño para que se posible escribir programas de forma sencilla. Los lenguajes de programación son independientes de la computadora, un programa escrito con este tipo de lenguajes es portable (tansportable), sin embargo es posible que algunos programas sean un poco dependientes del sistema operativo que se este utilizando, pero puede ser ejecutado con algunas o pocas modificaciones de acuerdo a la computadora y del sistema operativo utilizados [C]. En la actualidad existen una gran cantidad de lenguajes, la mayoría son especializados a un enfoque en particular, como por ejemplo C, Java, .Net, C#, C++, PHP, Prolog, Pascal, Haskell, Lisp, etc.

Características de los lenguajes de programación:

- Dentro de su sintaxis presentan similitud con el lenguaje natural.
- Una instrucción puede representar muchas instrucciones en un lenguaje de bajo nivel.
- Portabilidad entre computadoras, independiente del hardware y algunas veces del sistema operativo.
- Al tener un lenguaje de alto nivel se requiere de un tiempo considerable al efectuar las traducciones a un lenguaje máquina.
- Se requiere de suficiente memoria para ejecutar el programa en comparación de programas escritos en lenguajes de bajo nivel.
- Al programar en bajo nivel se tiene un mejor aprovechamiento de los recursos de la computadora, ya que interactúan con directamente con el sistema operativo o con la misma computadora.

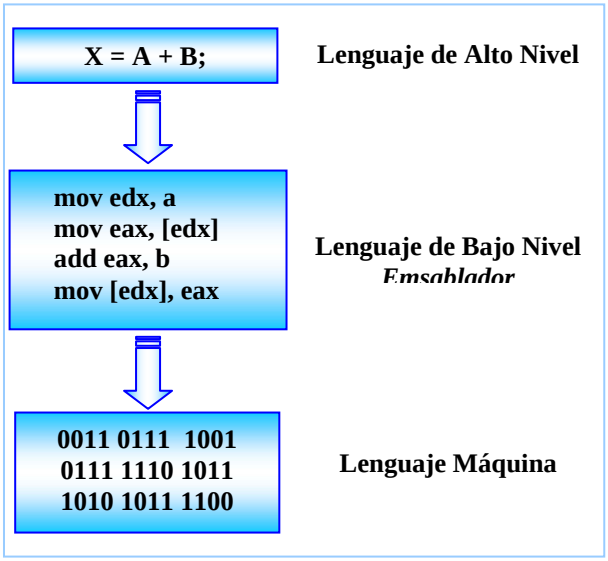


Figura 5.- Nivel de abstracción con los distintos niveles de Lenguajes de programación.

El programador hace uso de otros tipos de programas conocidos como **IDE** (Integrated Development Environment – **Entorno de Desarrollo Integrado**) con los cuales es posible desarrollar programas o aplicaciones de manera más rápida. Un IDE es un conjunto de programas o herramientas utilizados para facilitar a los programadores desarrollar aplicaciones, ya que comúnmente contienen un editor para codificar el programa, un compilador o un interprete, depurador, etc. (sección 2.3.1).Algunos ejemplos de IDE se encuentran los siguientes: Turbo C, Dev C++, Borland, Eclipse, NetBeans, JBuilder, etc.

Las computadoras están divididas en dos partes fundamentales, por una parte se tiene el **hardware** que representan la estructura en componentes físicos que son los dispositivos tangibles de la computadora, y por otro lados se tiene el **software** que representa a los programas y sistemas que son elementos no tangibles de la computadora.

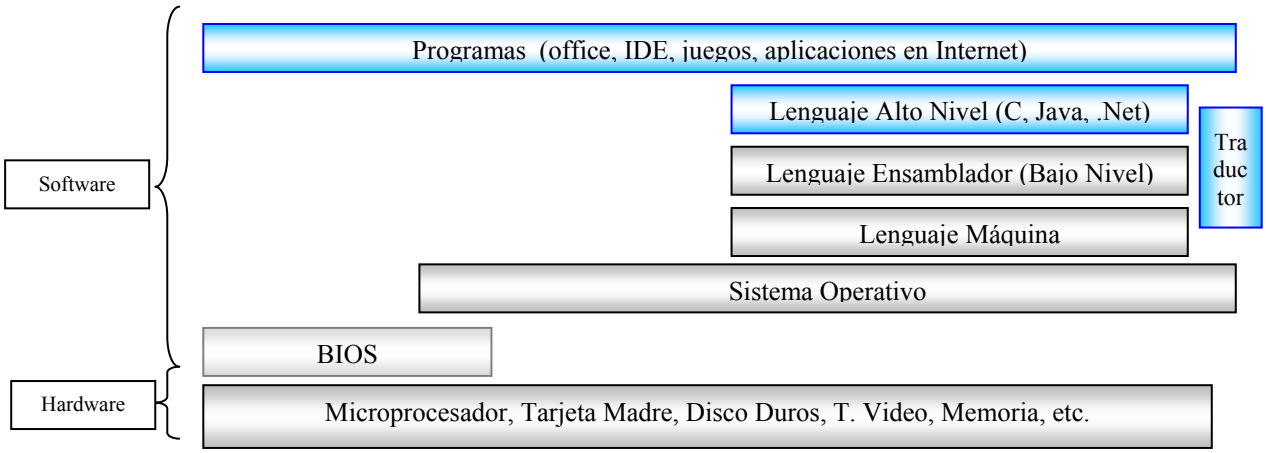


Figura 6.- Estructura de la computadora en hardware y software.

2.3.3 Traductores (Compiladores e Intérpretes)

Los traductores de lenguajes son programas que traducen programas fuentes escritos en lenguajes de alto nivel a código máquina. Los traductores se dividen en:

- **Intérpretes**: Son traductores que toman un programa o código fuente, los traduce y a continuación los ejecuta, este proceso se efectúa línea a línea, si se encuentra algún error se detienen el interprete [5]. Ejemplos de algunos intérpretes se encuentran Qbasic, QuickBASICK, Smalltalk, etc. En la siguiente figura se describen las fases de un intérprete.

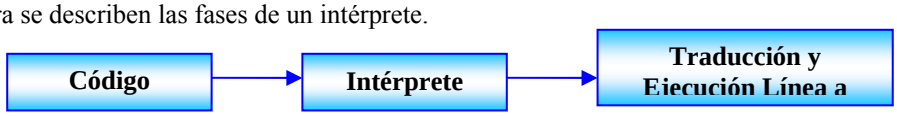


Figura 7.- Fase de un Intérprete.

- **Compilador**: Es un programa que traduce los programas o códigos fuentes escritos en lenguaje de alto nivel a lenguaje máquina. La traducción de un programa fuente se le llama ***programa objeto o código objeto***. El compilador traduce instrucción a instrucción del código fuente a código objeto, pero se efectúa en todo el programa objeto y si existen errores en el código fuente, estos se describen al final de la *compilación*. El código objeto es traducido a código máquina que será ejecutable. En la siguiente figura se describen las fases de compilación [5]. Algunos ejemplos de compiladores se encuentran: C, C++, Java, .Net, C#, Pascal, Cobol, etc.

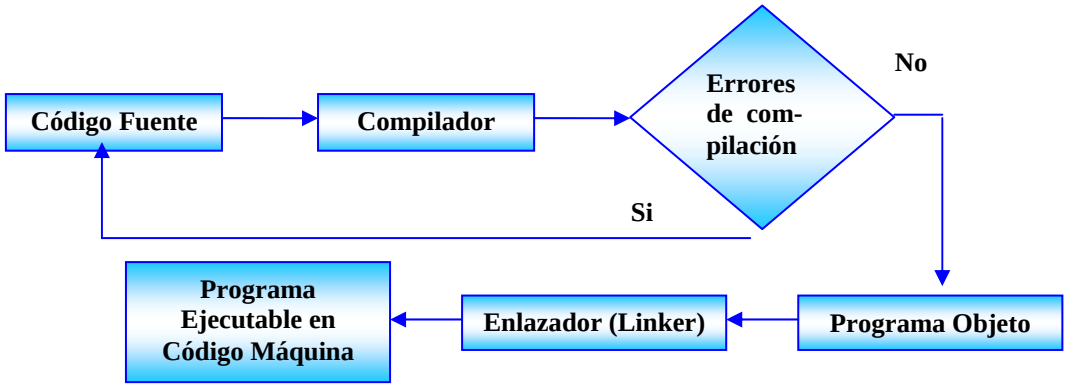


Figura 8.- Fase de un compilador.

Para efectuar la captura del código fuete se hace uso de un **editor de texto**, que puede ser uno de los programas de algún IDE. Al analizar el diagrama se hace dar a notar que mientras existan errores de compilación no es posible generar un programa objeto que permitiría obtener un programa ejecutable. Para la obtención del programa ejecutable en código máquina es necesario utilizar un enlazador, el cual genera el programa ejecutable [5].

Resumiendo del proceso de ejecución de un programa se presenta el siguiente diagrama que representa aspectos referentes a introducir el código fuente, llevar a cabo la ejecución, introducir datos y obtener resultados de la ejecución del proceso. También en dicho diagrama se ven claramente los tres aspectos a identificar de un algoritmo (sección 2.2)

Figura 9.- Codificación, compilación y ejecución de un programa.

2.4 Principios de Programación Modular

La **programación modular** es uno de los métodos de diseño más flexible y potente para mejorar la productividad de un programa. El objetivo es dividir el programa en ***módulos*** (partes independientes) que pueden ser desarrollados, es decir, analizado, codificado y ejecutados de manera independiente a los demás módulos, para posteriormente ser integrados a un programa único [5].

Un **módulo** se define como una serie de pasos o instrucciones (conocidas como rutinas) que se denominan funciones o procedimientos.

La descomposición de un programa en módulos independientes más simples se conoce también como el método de "**Divide y Vencerás** (divide and conquer)". El desarrollo de cada uno de los módulos se efectúa de forma independiente, y siguiendo un método descendente o ascendente se llegará hasta la descomposición final del problema (que representará el programa) en módulos en forma jerárquica [5].

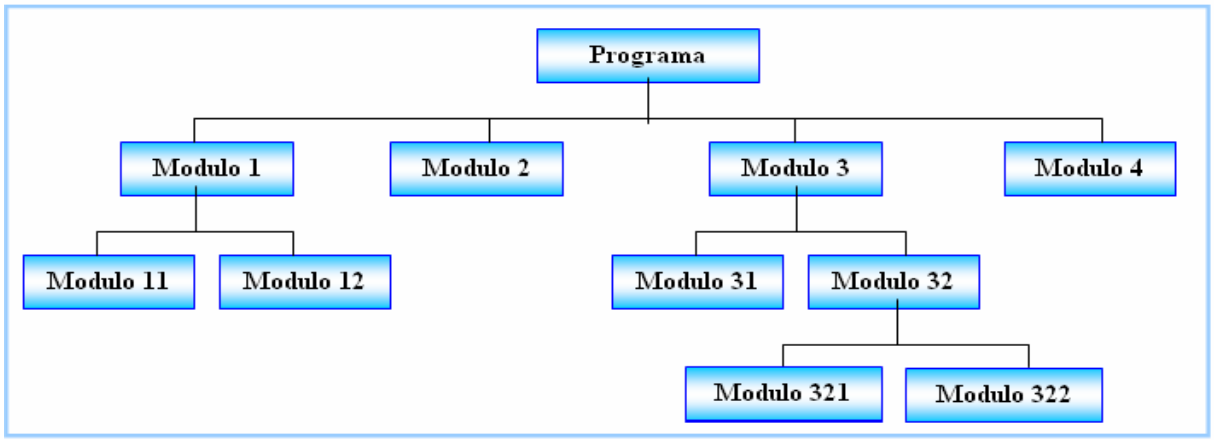


Figura 10.- Programación modular.

2.4.1 Diseño Descendente (Top-Down)

El diseño descendente es el proceso mediante el cual el problema se descompone en una serie de niveles o pasos sucesivos de refinamiento. Se efectúa la descomposición del problema en etapas o estructuras jerárquicas, de forma que se puede considerar cada estructura desde dos puntos de vista ***¿qué hace?*** y ***¿cómo lo hace?***

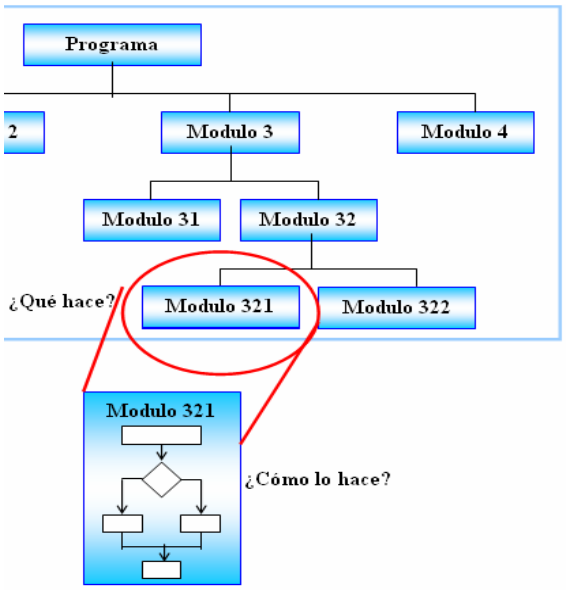


Figura 11.- Diseño descendente. ¿Qué hace? y ¿Cómo lo hace?

2.5 Ciclos de Vida de Software - Modelo en cascada

Existen dos niveles de construcción de programas: pequeños que son realizados de manera ***individual por el programador*** y los grandes que representan sistemas (conocidos como **proyectos de software**) que son realizados por un ***equipo de programadores*** [5].

Para el desarrollo de proyectos de software se utilizan metodologías, procesos o métodos que permiten desarrollar de manera disciplinada el proyecto. Cada uno de estos métodos se enfoca en el **ciclo de vida del software** en el cual se desarrollar primero un **análisis** del problema, se efectúa un **diseño**, se desarrolla la **codificación** (obtención del código fuente), se verifica la **compilación** para **depurar** errores y finalmente se desarrolla la mejora o **mantenimiento** del programa.

El **modelo en cascada** permite representar y ordenar las etapas del ciclo de vida de software de forma disciplinada, ya que se debe de completar una etapa para continuar con la siguiente etapa del modelo. En la siguiente figura se muestra cada una de estas etapas y el orden que deben de seguirse en el desarrollo del ciclo de vida del software.

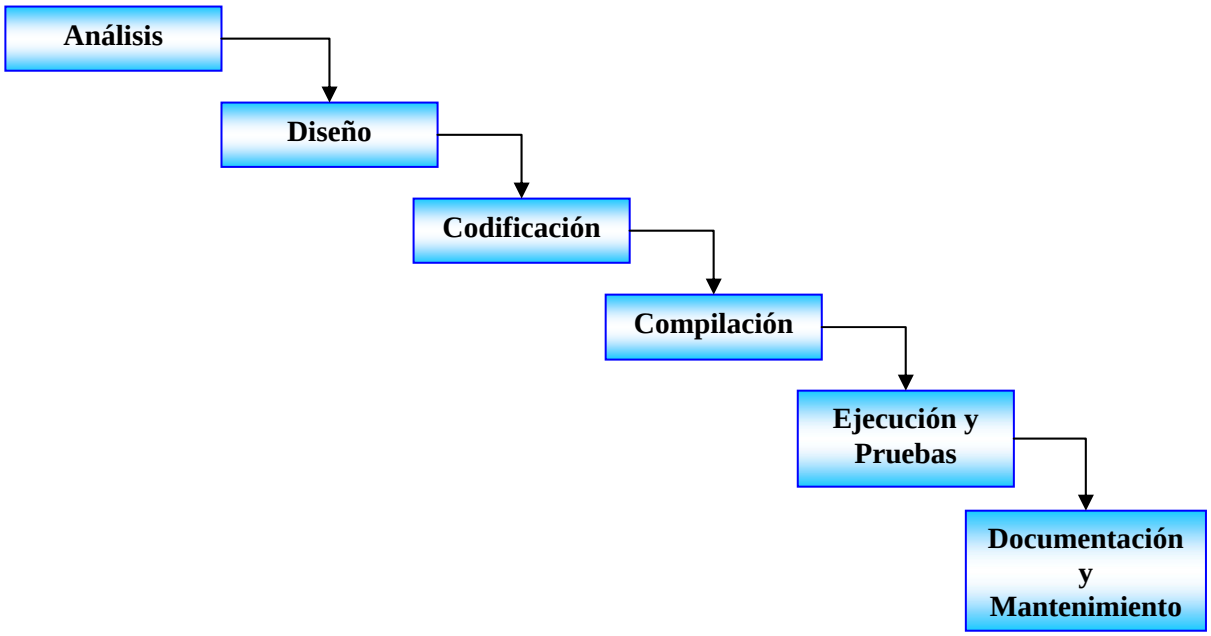


Figura 12.- Modelo en cascada

2.5.1 Análisis

En esta primera etapa se debe de determinar ***qué es*** los que el programa deberá de hacer. Se deben de plantear y contestar las siguientes preguntas:

- ¿Cuáles son los datos (la información) de entrada? (**Entrada**)
- ¿Cuáles son los resultados esperados? (**Salida**)

La información de entrada es la que el usuario de forma directa o indirecta proporcionará al programa, mientras que los resultados esperados son los que se obtendrán después de ejecutar uno o más procesos dentro del programa.

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"
Entrada: Temperatura en grados Celsius (C).
Salida: Temperatura en grados Fahrenheit (F).

2.5.2 Diseño

Una vez que la etapa de análisis se haya finalizado se efectúa la etapa de diseño. El objetivo de la etapa de diseño es determinar ***cómo*** debe de hacerse el proceso. Para describir el cómo debe de desarrollarse generalmente se describen con el uso de **algoritmos** (sección 2.2) que representan el procedimiento.

- ¿Cómo se efectúa el proceso? (**Procedimiento**)

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"
Entrada: Temperatura en grados Celsius (C).
Salida: Temperatura en grados Fahrenheit (F).
Procedimiento:
1.- Determinar el valor de la temperatura en Celsius C.
2.- Desarrollar la formula 9/5 * C + 32 y asignar el resultado a F.
3.- Indicar el resultado de la conversión contenido en F.

Para desarrollar la fase de diseño se hace uso de dos mecanismos:
- Diagramas de flujo.
- Pseudocódigo.

Un **diagrama de flujo** representa un algoritmo de manera grafica haciendo uso de diagramas que utilizan símbolos (cajas) de manera estandarizada [5] y que permiten representa los pasos de un algoritmo considerando las tres características del algoritmo. La secuenciación se determina mediante flechas que unen los símbolos (cajas) e indica la **línea de flujo** de ejecución del algoritmo.

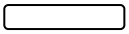
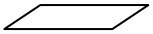
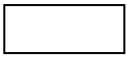
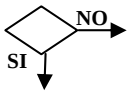
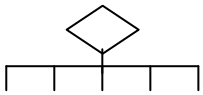
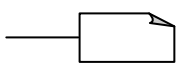
Símbolo	Función
	***Terminal*** – Representa el inicio o fin de un programa.
	***Entrada/Salida*** – Representa interacción con el usuario, para entrada o para salida.
	***Proceso*** – Cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, etc.
	***Puntos de decisión*** – Indica operaciones lógica o de comparación entre datos y en función del resultado se determina cual de los dos distintos caminos alternativos se efectuará, ya sea sí se cumple la condición (SI) o en otro caso no se cumpla (NO).
	***Conector*** – Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector de salida y otro de entrada. Se refiere al conector dentro de una misma página.
	Indicador de dirección o línea de flujo – Indica el sentido de ejecución del programa.
	Línea conectora – Sirve de unión entre dos símbolos.
	***Decisión múltiple*** – En base a la condición de comparación se seguirá uno y solo uno de los posibles caminos de acuerdo a resultado obtenido.
	***Comentarios*** – Utilizado para añadir comentarios a los símbolos del diagrama de flujo, se puede dibujar en cualquier parte que no interfiera con el diagrama de flujo.

Tabla 1.- Símbolos del diagrama de flujo [5].

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"

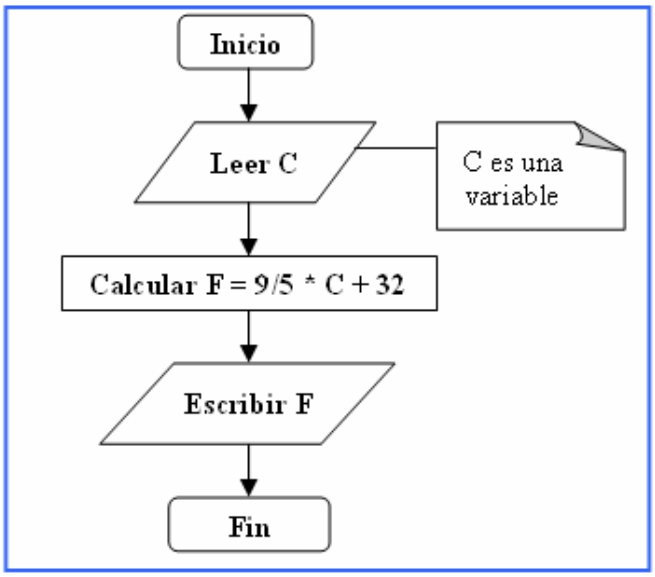


Figura 13.- Ejemplo de un diagrama de flujo.

El **pseudocódigo** es un lenguaje cercano al lenguaje de programación de alto nivel, tiene como objetivo describir la ejecución del programa utilizando un lenguaje con una sintaxis y semántica definida y estandarizada.

En el pseudocódigo no se considera aspectos específicos a un lenguaje de programación, es decir, es ***independiente del lenguaje de programación*** a seleccionar en la siguiente etapa. Y al elaborar un pseudocódigo bien estructurado permite obtener fácilmente la traducción a un lenguaje seleccionado como puede ser: C, Java, .Net. etc.

Permite concentrarse en la lógica y estructura del control y no en el desarrollo de un lenguaje en específico. En programas grandes es conveniente tener un bosquejo del programa en diagrama de flujo, pero es necesario tener el pseudocódigo completo.

Es posible utilizar palabras en un idioma particular, sin embargo es necesario recalcar que existen **palabras reservadas** (aquellas que son utilizados para definir el pseudocódigo y no pueden ser utilizadas por el programador para evitar conflictos de nombrado) que son marcada con el tipo de letra en negrita.

Existen reglas que permiten hacer legible el pseudocódigo, algunas de las más importantes son respetar el ***identado*** y tener una buena ***documentación***.

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"

Pseudocódigo "***ConversiónCF***"

```
inicio
        leer ( C )
        F ← (9/5)C+32
        escribir ( C )
fin
```

2.5.3 *Codificación (implementación)*

En esta etapa se efectúa la traducción de los algoritmos (del pseudocódigo o del diagrama de flujo) en un programa escrito en un lenguaje de programación en particular. La mayoría de veces es más sencillo traducir a partir del pseudocódigo obtenido en la fase de diseño a un código fuente utilizando una especie de ***diccionario***.

Las reglas básicas para tener una codificación adecuada son tener **identado** el código y generar una buena ***documentación***.

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"

```
#include <stdio.h>

int main()
{
   int C = 0 ;
   int F = 0 ;
   printf(« Teclee la temperatura en Celcius : ») ;
   scanf(« %d », &C) ;

   F= 9/5*C + 32 ;

   printf("\n La temperatura en grados Fahrenheit es: %d", F);

   getch( );

   return (0);
}
```

2.5.4 *Compilación*

En la etapa de **compilación** se efectúa el proceso de traducción del ***código fuente*** a un ***programa ejecutable***, para esto se requiere de una herramienta llamada **compilador** (sección 2.3.3).

El compilador se encarga de verificar que el código fuente efectuando un análisis sintáctico, lexicográfico y semántico. En caso de que el compilador encuentre errores, es necesario que el programador los resuelva para continuar con la siguiente etapa. Una vez que el código este libre de errores se obtienen un archivo ejecutable que permite efectuar la ejecución y desarrollar las pruebas en la siguiente fase.

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"

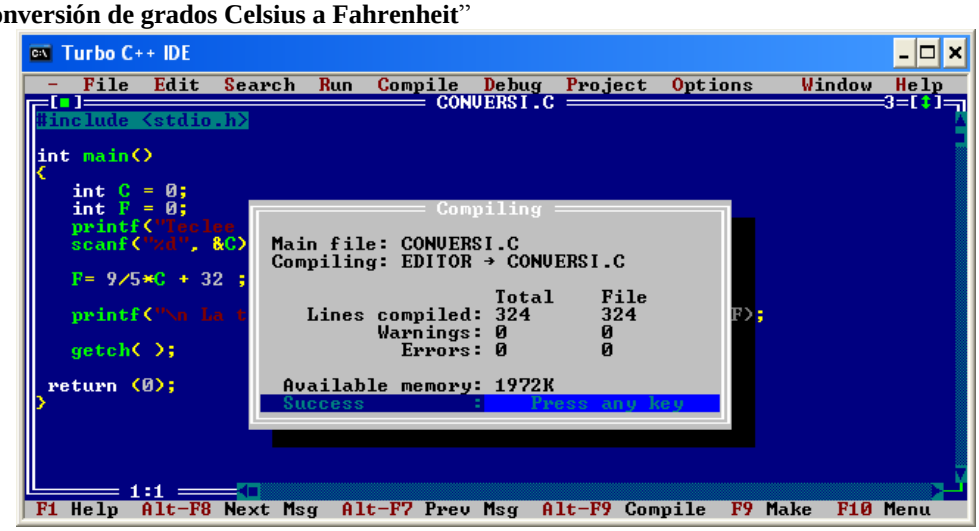


Figura 14.- Captura de pantalla del IDE Turbo C al compilar.

2.5.5 Ejecución y pruebas

Se requiere **verificar el funcionamiento del programa** que al ser ejecutado genere los resultados esperados; estos resultados (o salidas) deben de ser los mismos que se definieron en la fase de análisis.

Ejemplo "**Conversión de grados Celsius a Fahrenheit**"

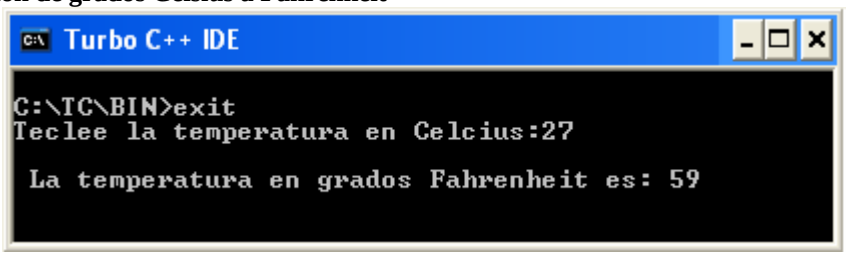


Figura 15.- Ejecución de un programa con Turbo C++

2.5.6 Documentación y Mantenimiento

La **documentación** de programas es tan importante como el mismo programa, puede ser reprensada mediante enunciados dentro del código o con el uso de documentos en donde es posible incluir diagramas o imágenes representativas. La documentación se tratara en la unidad 7.

El objetivo de la documentación es apoyar en el entendimiento del funcionamiento interno del programa y facilitar el mantenimiento del mismo. La documentación se divide en dos partes:

- Documentación interna
- Documentación externa.

El **mantenimiento** contempla la posibilidad efectuar futuras modificaciones en el programa, considerando la integración de nuevas funcionalidades o cambios en los datos de entrada o de salida.