**Nombre del alumno:** José Carlos Toledo Pérez

**Nombre del tema:** Unidad 1 y 2

**Parcial:** 1

**Nombre de la materia**: Fundamentos y lógica de programación

**Nombre del profesor:** Juan José Ojeda Trujillo

**Nombre de la Licenciatura:** Ingeniería en Sistemas Computacionales

**Cuatrimestre:** 3

Unidad 1 INTRODUCCIÓN A LA PROGRAMACIÓN

1.1 Lenguajes de programación

1.1.1 Traductores de lenguaje: proceso de traducción de un programa

1.1.2 La compilación y sus fases

1.2 Evolución de los lenguajes de programación

1.3 Paradigmas de programación

1.4 Historia de los lenguajes de programación

1.5 Fases en la resolución de problemas

1.6 Análisis del problema

1.7 Diseño de algoritmo

1.8 Herramientas de programación

1.9 Codificación de un programa

1.10 Compilación y ejecución de un programa

1.11 Verificación y depuración de un programa

Unidad 2 TIPOS DE PROGRAMACIÓN Y ALGORITMOS

2.1 Programación modular

2.2 Programación estructurada

2.3 Programación orientada a objetos

2.4 Abstracción

2.5 Encapsulación y ocultación de datos

2.6 Objetos

2.7 Clases

2.8 Generalización y especialización: herencia

2.9 Reusabilidad

2.10 Polimorfismo

2.11 Algoritmos

2.11.1 Concepto y características de los algoritmos

2.11.2 Diseño del algoritmo

2.11.3 Estructura de algoritmos

2.11.4 Representación gráfica de los algoritmos

2.11.5 Pseudocódigo

2.11.6 Diagrama de flujo

## 1.1 Lenguajes de programación

En la realidad la computadora no entiende directamente los lenguajes de programación, sino que se requiere un programa que traduzca el código fuente a otro lenguaje que sí entiende la máquina directamente, pero muy complejo para las personas; este lenguaje se conoce como lenguaje máquina y el código correspondiente código máquina.

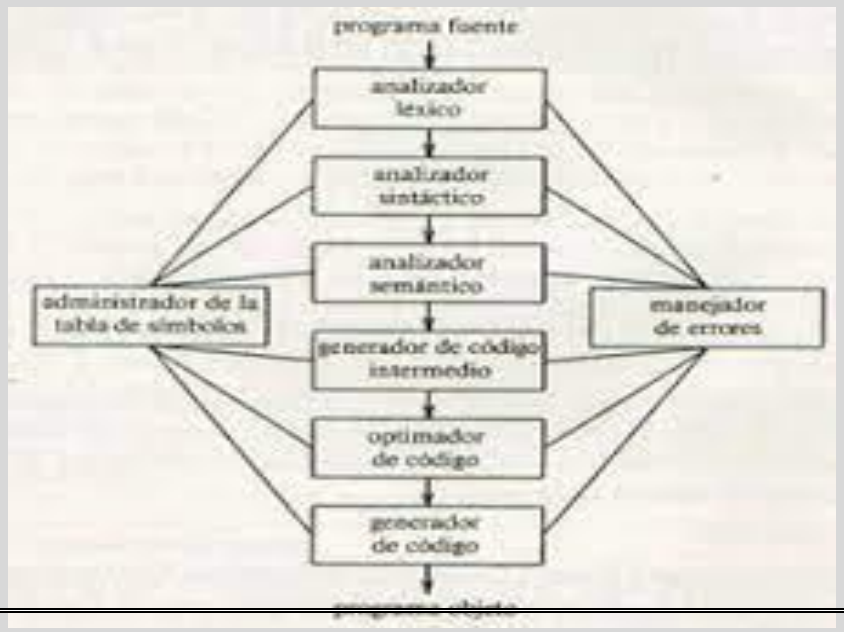
Los programas que traducen el código fuente escrito en un lenguaje de programación —tal como C++— a código máquina se denominan traductores. El proceso de conversión de un algoritmo escrito en pseudocódigo hasta un programa ejecutable comprensible por la máquina.

Hoy en día, la mayoría de los programadores emplean lenguajes de programación como C++, C, C#, Java, Visual Basic, XML, HTML, Perl, PHP, JavaScript..., aunque todavía se utilizan, sobre todo profesionalmente, los clásicos COBOL, FORTRAN, Pascal o el mítico BASIC. Estos lenguajes se denominan lenguajes de alto nivel y permiten a los profesionales resolver problemas convirtiendo sus algoritmos en programas escritos en alguno de estos lenguajes de programación.

### 1.1.1 Traductores de lenguaje: proceso

de traducción de un programa El proceso de traducción de un programa fuente escrito en un lenguaje de alto nivel a un lenguaje máquina comprensible por la computadora, se realiza mediante programas llamados “traductores”. Los traductores de lenguaje son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina.

1. Escritura del programa fuente con un editor (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento (por ejemplo, un disco).
2. Introducir el programa fuente en memoria.
3. Compilar el programa con el compilador seleccionado.
4. Verificar y corregir errores de compilación (listado de errores).
5. Obtención del programa objeto.
6. El enlazador (linker) obtiene el programa ejecutable.
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.



## 1.2 Evolución de los lenguajes de programación

En la década de los cuarenta cuando nacían las primeras computadoras digitales el lenguaje que se utilizaba para programar era el lenguaje máquina que traducía directamente el código máquina (código binario) comprensible para las computadoras. Las instrucciones en lenguaje máquina dependían de cada computadora y debido a la dificultad de su escritura, los investigadores de la época simplificaron el proceso de programación desarrollando sistemas de notación en los cuales las instrucciones se representaban en formatos nemónicos (nemotécnicos) en vez de en formatos numéricos que eran más difíciles de recordar.

## 1.3 Paradigmas de programación

La evolución de los lenguajes de programación ha ido paralela a la idea de paradigma de programación: enfoques alternativos a los procesos de programación. En realidad, un paradigma de programación representa fundamentalmente enfoques diferentes para la construcción de soluciones a problemas y por consiguiente afectan al proceso completo de desarrollo de software. Los paradigmas de programación clásicos son: procedimental (o imperativo), funcional, declarativo y orientado a objetos.

## 1.4 Historia de los lenguajes de programación

La historia de la computación ha estado asociada indisolublemente a la aparición y a la historia de lenguajes de programación de computadoras26. La Biblia de los lenguajes ha sido una constante en el desarrollo de la industria del software y en los avances científicos y tecnológicos. Desde el año 1642 en que Blaise Pascal, inventó La Pascalina, una máquina que ayudaba a contar mediante unos dispositivos de ruedas, se han sucedido numerosos inventos que han ido evolucionando, a medida que se programaban mediante códigos de máquina, lenguajes ensambladores, hasta llegar a los lenguajes de programación de alto nivel en los que ya no se dependía del hardware de la máquina sino de la capacidad de abstracción del programador y de la sintaxis, semántica y potencia del lenguaje.

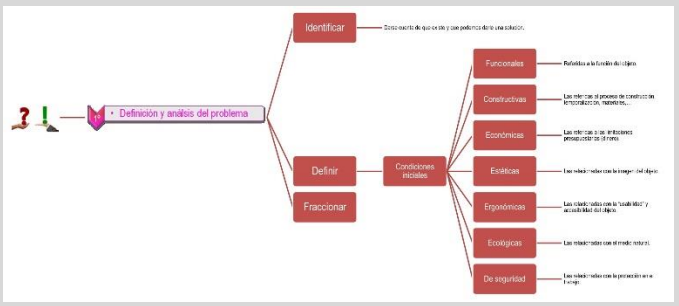
## 1.5 Fases en la resolución de problemas

Las fases de resolución de un problema con computadora son:

- Análisis del problema.
- Diseño del algoritmo.
- Codificación.
- Compilación y ejecución.
- Verificación.
- Depuración.
- Mantenimiento.
- Documentación.

## 1.6 Análisis del problema

La primera fase de la resolución de un problema con computadora es el análisis del problema. Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada. Dado que se busca una solución por computadora, se precisan especificaciones detalladas de entrada y salida. La Figura muestra los requisitos que se deben definir en el análisis




## 1.7 Diseño de algoritmo

En la etapa de análisis del proceso de programación se determina qué hace el programa. En la etapa de diseño se determina cómo hace el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocido divide y vencerás. Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividiendo estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora.

## 1.8 Herramientas de programación

Las dos herramientas más utilizadas comúnmente para diseñar algoritmos son: diagramas de flujo y pseudocódigos. Un diagrama de flujo (flowchart) es una representación gráfica de un algoritmo. Los símbolos utilizados han sido normalizados por el Instituto Norteamericano de Normalización (ANSI), y los más frecuentemente empleados se muestran en la Figura 2.2, junto con una plantilla utilizada para el dibujo de los diagramas de flujo (Figura 2.3). En la Figura 2.4 se representa el diagrama de flujo que resuelve el Problema.

El pseudocódigo es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

1.9 Codificación de un programa La codificación es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con igual facilidad en un lenguaje o en otro. Para realizar la conversión del algoritmo en programa se deben sustituir las palabras reservadas en español por sus homónimos en inglés, y las operaciones/instrucciones indicadas en lenguaje natural por el lenguaje de programación correspondiente.

1.10 Compilación y ejecución de un programa

Una vez que el algoritmo se ha convertido en un programa fuente, es preciso introducirlo en memoria mediante el teclado y almacenarlo posteriormente en un disco. Esta operación se realiza con un programa editor. Posteriormente el programa fuente se convierte en un archivo de programa que se guarda (graba) en disco. El programa fuente debe ser traducido a lenguaje máquina, este proceso se realiza con el compilador y el sistema operativo que se encarga prácticamente de la compilación. Si tras la compilación se presentan errores (errores de compilación) en el programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo.

1.11 Verificación y depuración de un programa La verificación o compilación de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados datos de test o prueba, que determinarán si el programa tiene o no errores (“bugs”). Para realizar la verificación se debe desarrollar una amplia gama de datos de test: valores normales de entrada, valores extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa.

La depuración es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores.

Cuando se ejecuta un programa, se pueden producir tres tipos de errores:

1. Errores de compilación

2. Errores de ejecución.

3. Errores lógicos.

## 1.12 Documentación y mantenimiento

La documentación de un programa puede ser interna y externa. La documentación interna es la contenida en líneas de comentarios. La documentación externa incluye análisis, diagramas de flujo y/o pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.

La documentación es vital cuando se desea corregir posibles errores futuros o bien cambiar el programa. Tales cambios se denominan mantenimiento del programa. Después de cada cambio la documentación debe ser actualizada para facilitar cambios posteriores.

# Unidad II TIPOS DE PROGRAMACIÓN Y ALGORITMOS

## 2.1 Programación modular

La programación modular es uno de los métodos de diseño más flexible y potente para mejorar la productividad de un programa. En programación modular el programa se divide en módulos (partes independientes), cada uno de los cuales ejecuta una única actividad o tarea y se codifican independientemente de otros módulos. Cada uno de estos módulos se analiza, codifica y pone a punto por separado. Cada programa contiene un módulo denominado programa principal que controla todo lo que sucede; se transfiere el control a submódulos (posteriormente se denominarán subprogramas), de modo que ellos puedan ejecutar sus funciones; sin embargo, cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea

## 2.2 Programación estructurada

C, Pascal, FORTRAN, y lenguajes similares, se conocen como lenguajes procedimentales (por procedimientos). Es decir, cada sentencia o instrucción señala al compilador para que realice alguna tarea: obtener una entrada, producir una salida, sumar tres números, dividir por cinco, etc. En resumen, un programa en un lenguaje procedimental es un conjunto de instrucciones o sentencias. En el caso de pequeños programas, estos principios de organización (denominados paradigma) se demuestran eficientes. El programador sólo tiene que crear esta lista de instrucciones en un lenguaje de programación, compilar en la computadora y ésta, a su vez, ejecuta estas instrucciones.

## 2.3 Programación orientada a objetos La idea fundamental de los lenguajes orientados a objetos es combinar en una única unidad o módulo, tanto los datos como las funciones que operan sobre esos datos. Tal unidad se llama un objeto. Las funciones de un objeto se llaman funciones miembros en C++ o métodos (éste es el caso de Smalltalk, uno de los primeros lenguajes orientados a objetos), y son el único medio para acceder a sus datos. Los datos de un objeto, se conocen también como atributos o variables de instancia. Si se desea leer datos de un objeto, se llama a una función miembro del objeto.

## 2.4 Abstracción

La abstracción es la propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista determinado y no tener en cuenta los restantes aspectos. El término abstracción que se suele utilizar en programación se refiere al hecho de diferenciar entre las propiedades externas de una entidad y los detalles de la composición interna de dicha entidad. Es la abstracción la que permite ignorar los detalles internos de un dispositivo complejo tal como una computadora, un automóvil, una lavadora o un horno de microondas, etc., y usarlo como una única unidad comprensible.

## 2.5 Encapsulación y ocultación de datos

El encapsulado o encapsulación de datos es el proceso de agrupar datos y operaciones relacionadas bajo la misma unidad de programación. En el caso de los objetos que poseen las mismas características y comportamiento se agrupan en clases, que no son más que unidades o módulos de programación que encapsulan datos y operaciones.

## 2.6 Objetos

El objeto es el centro de la programación orientada a objetos. Un objeto es algo que se visualiza, se utiliza y juega un rol o papel. Si se programa con enfoque orientado a objetos, se intentan descubrir e implementar los objetos que juegan un rol en el dominio del problema y en consecuencia programa. La estructura interna y el comportamiento de un objeto, en una primera fase, no tiene prioridad.

## 2.7 Clases En POO los objetos son miembros de clases. En esencia, una clase es un tipo de datos al igual que cualquier otro tipo de dato definido en un lenguaje de programación. La diferencia reside en que la clase es un tipo de dato que contiene datos y funciones. Una clase contiene muchos objetos y es preciso definirla, aunque su definición no implica creación de objetos.

Una clase es, por consiguiente, una descripción de un número de objetos similares. Madonna, Sting, Prince, Juanes, Carlos Vives o Juan Luis Guerra son miembros u objetos de la clase "músicos de rock". Un objeto concreto, Juanes o Carlos Vives, son instancias de la clase "músicos de rock".

## 2.8 Generalización y especialización: herencia

La generalización es la propiedad que permite compartir información entre dos entidades evitando la redundancia. En el comportamiento de objetos existen con frecuencia propiedades que son comunes en diferentes objetos y esta propiedad se denomina generalización.

## 2.9 Reusabilidad

Una vez que una clase ha sido escrita, creada y depurada, se puede distribuir a otros programadores para utilizar en sus propios programas. Esta propiedad se llama reusabilidad 4 o reutilización. Su concepto es similar a las funciones incluidas en las bibliotecas de funciones de un lenguaje procedimental como C que se pueden incorporar en diferentes programas.

En C++, el concepto de herencia proporciona una extensión o ampliación al concepto de reusabilidad. Un programador puede considerar una clase existente y sin modificarla, añadir competencias y propiedades adicionales a ella. Esto se consigue derivando una nueva clase de una ya existente. La nueva clase heredará las características de la clase antigua, pero es libre de añadir nuevas características propias.

## 2.10 Polimorfismo

Polimorfismo es la propiedad de que un operador o una función actúen de modo diferente en función del objeto sobre el que se aplican. En la práctica, el polimorfismo significa la capacidad de una operación de ser interpretada sólo por el propio objeto que lo invoca. Desde un punto de vista práctico de ejecución del programa, el polimorfismo se realiza en tiempo de ejecución ya que durante la compilación no se conoce qué tipo de objeto y por consiguiente qué operación ha sido llamada.

## 2.11 Algoritmos

El objetivo fundamental de este texto es enseñar a resolver problemas mediante una computadora. El programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. A lo largo de todo este libro nos referiremos a la metodología necesaria para resolver problemas mediante programas, concepto que se denomina metodología de la programación. El eje central de esta metodología es el concepto, ya tratado, de algoritmo.

### 2.11.1 Concepto y características de los algoritmos

Un algoritmo es un método para resolver un problema. El profesor Niklaus Wirth —inventor de Pascal, Modula-2 y Oberon— tituló uno de sus más famosos libros, Algoritmos + Estructuras de datos = Programas, significándonos que sólo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos. Esta ecuación será una de las hipótesis fundamentales consideradas en esta obra.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto.

### 2.11.2 Diseño del algoritmo

Una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la máquina constituyen, como ya conocemos, el algoritmo. La información proporcionada al algoritmo constituye su entrada y la información producida por el algoritmo constituye su salida.

### 2.11.3 Escritura de algoritmos

Como ya se ha comentado anteriormente, el sistema para describir ("escribir") un algoritmo consiste en realizar una descripción paso a paso con un lenguaje natural del citado algoritmo. Recordemos que un algoritmo es un método o conjunto de reglas para solucionar un problema. En cálculos elementales estas reglas tienen las siguientes propiedades:

• deben ir seguidas de alguna secuencia definida de pasos hasta que se obtenga un resultado coherente, • sólo puede ejecutarse una operación a la vez.

### 2.11.4 Representación gráfica de los algoritmos

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Ello permitirá que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje.

Para conseguir este objetivo se precisa que el algoritmo sea representado gráfica o numéricamente, de modo que las sucesivas acciones no dependan de la sintaxis de ningún lenguaje de programación, sino que la descripción pueda servir fácilmente para su transformación en un programa, es decir, su codificación.

Los métodos usuales para representar un algoritmo son:

1. diagrama de flujo
2. diagrama N-S (Nassi-Schneiderman)
3. lenguaje de especificación de algoritmos: pseudocódigo
4. lenguaje español, inglés…
5. fórmulas.

### 2.11.5 Pseudocódigo

El pseudocódigo es un lenguaje de especificación (descripción) de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil. Los lenguajes APL Pascal y Ada se utilizan a veces como lenguajes de especificación de algoritmos.

2.11.6 Diagrama de flujo

Un diagrama de flujo (flowchart) es una de las técnicas de representación de algoritmos más antigua y a la vez más utilizada, aunque su empleo ha disminuido considerablemente, sobro todo, desde la aparición de lenguajes de programación estructurados. Un diagrama de flujo es un diagrama que utiliza los símbolos (cajas) estándar y que tiene los pasos de algoritmo escritos en esas cajas unidas por flechas, denominadas líneas de flujo, que indican la secuencia en que se debe ejecutar.

Fuentes Bibliográficas:

https://plataformaeducativauds.com.mx/assets/docs/libro/ISC/3651f6209f73201727db65c8fc768c9c-LC-ISC304%20FUNDAMENTOS%20Y%20L%C3%93GICA%20DE%20PROGRAMACI%C3%93N.pdf