UNIVERSIDAD NACIONAL DE EDUCACIÓN

Enrique Guzmán y Valle
Alma Máter del Magisterio Nacional

FACULTAD DE CIENCIAS

Escuela Profesional de Matemática e Informática

MONOGRAFÍA

ELEMENTOS DE PROGRAMACIÓN

Algoritmos, herramientas de Algoritmos, programación estructurada: C++, C Sharp, estructura de datos, cadenas de caracteres, tipos de datos, procedimientos y funciones, aplicaciones.

Examen de Suficiencia Profesional Res. N°0519-2018-D-FAC

Presentada por:

Roger Ostos Acevedo

Para optar al Título Profesional de Licenciado en Educación

Especialidad: Matemática e Informática

Lima, Perú

2018

MONOGRAFÍA

ELEMENTOS DE PROGRAMACIÓN

Algoritmos, herramientas de algoritmos, programación estructurada: C++, C Sharp, estructura de datos, cadenas de caracteres, tipos de datos, procedimientos y funciones, aplicaciones.

Designación de Jurado Resolución Nº0519-2018-D-FAC

Mg. William Alberto HUAMANI ESCOBAR

Presidente

Dr. Adrián QUISPE ANDIA

Secretaria

Dr. Guillermo Pastor MORALES ROMERO

Vocal

Línea de Investigación: Lengua y comunicación en el campo educativo.

Dedicatoria

A mis padres y a los Maestro de la de la Universidad Nacional Enrique Guzmán y Valle (CANTUTA) por su enseñanza y dedicación hacia los alumnos.

Tabla de contenido

Introducción

Existen muchos lenguajes de programación, han evolucionado de tal manera que cada vez son más sencillos de utilizar. Desde la programación de caída libre hasta la actual orientada a objetos cada vez es más importante su aprendizaje y no solo por estudiantes de informática sino por todos los que alguna vez utilizaran las computadoras.

Los lenguajes de programación de alto nivel solucionaron la complejidad de realizar código, ya que se tenía que conocer el lenguaje de máquina y como sabemos la computadora como maquina electrónica para que funcione emplea el lenguaje binario (0 y 1).

Los dispositivos que utilizan, pueden almacenar y procesar información representada como variable. Este está representado por el procesador que cada día son más rápidos y de mejor performance de tal manera que los tiempos de procesos van mejorando por lo que se requiere que los algoritmos sean eficientes.

Capítulo I: Algoritmos

1.1 Algoritmos

> Un algoritmo es un conjunto de operaciones y procedimientos que deben seguirse para resolver un problema. La palabra algoritmo se deriva del nombre latinizado del gran Matemático Árabe Mohamed Ibn Al Kow Rizmi, el cual escribió sobre los años 800 y 825 su obra Quitad Al Mugabala, donde se recogía el sistema de numeración hindú y el concepto del cero. Fue Fibonacci, el que tradujo la obra al latín y el inicio con la palabra: Algoritmi Dicit. (Joyanes, 2015, p. 12)

El algorítmico permite realizar un análisis previo del problema a resolver y encontrar un método que permita resolverlo. Se dice que también es un conjunto de procedimientos que permite solucionar problemas mediante datos precisos, definidos y finitos.

Se tiene primeramente que diseñar el algoritmo, y luego codificarlo en el lenguaje de programación seleccionada.

De acuerdo a Joyanes:

> Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que lo ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizaran sin importar el idioma del cocinero. (Joyanes, 2015, p. 12)

Los algoritmos son el punto de partida de cualquier concepción para resolver un problema de resolución de problemas de procesamiento de datos. Un lenguaje de programación expresa el algoritmo y la maquina mediante el procesador lo procesa para ejecutarlo.

1.2 Representación de Algoritmos

Los algoritmos pueden ser elaborados de diferentes formas, con el lenguaje natural, por medio de pseudocódigo, utilizando diagramas de flujo y lenguajes de programación. El empleo de pseudocódigo y diagramas de flujo nos permite agilizar y evitar ambigüedades del lenguaje natural. Estas son formas más estructuradas que nos permiten la representación de algoritmos; sin embargo, son independientes del lenguaje de programación.

Un algoritmo usualmente se hace en tres niveles:

1. Descripción de alto nivel. Se define y reconoce el problema para lo cual se define el modelo matemático a seguir.
2. Descripción formal. Se describen los pasos que permiten resolver el problema.

3. Implementación. El algoritmo es codificado en el lenguaje de programación seleccionado.

1.2.1 Pseudocódigos

Se pueden emplear ordinogramas para representar gráficamente un problema pero hay problemas con el espacio. Cuando los algoritmos crecen, la resolución de problemas da lugar a ordinogramas tan grandes que deberán ser fragmentados, lo cual va dificultando la interpretación de estos.

Por tanto es esencial, el Pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. Es decir es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El Pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo:

- Ocupa menos espacio
- Permite representar de manera sencilla procesos que son repetitivas y muy complejas.
- Fácil para pasar de un Pseudocódigo al código de algún lenguaje de programación.
- Cumpliendo estas reglas se observa facilmente los niveles que tienen las operaciones.

Un ejemplo de algoritmo es el que realizamos para cambiar una llanta a un automóvil:
Inicio

Aflojar tornillos de las llantas

Levantar el coche con el gato

Sacar los tornillos de las llantas

Retirar la llanta

Colocar el repuesto

Atornillarlos

Bajar el coche con el gato

Ajustar los tornillos

Fin

1.2.2 Diagrama de Flujo

Los diagramas de flujos son representaciones gráficas de algoritmos, en forma detallada de cómo se procesa cada paso en la computadora para resolver el prblema.

Esta representación gráfica se da cuando varios símbolos, se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar los procesos. Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI):

Reglas para la construcción del Diagrama de Flujo:

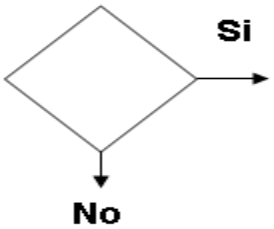
- Indicar el inicio y el final de nuestro diagrama de flujo.
- Las líneas usadas deben ser de flujo horizontales y/o verticales. No pueden ser inclinadas ni cruzadas.
- Las líneas de conexión deben ser conectada a un símbolo y no a varios.
- Poner comentarios que ayude a entender
- Sólo cuando sea necesario se utilizan los conectores.
- Todas las líneas de flujo tienen que estar conectadas.

- Se lee de arriba hacia abajo y de izquierda a derecha.
- El texto escrito en un símbolo deberá ser claro y preciso.

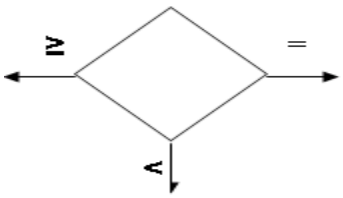
Los símbolos del diagrama de flujo tienen significado especial.

Inicio o Fin de un Algoritmo.

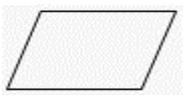
Representa una operación o una asignación.



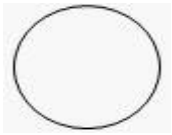
Representa decisiones alternativas bajo una expresión lógica.



Expresa decisiones alternativas de una expresión numérica.

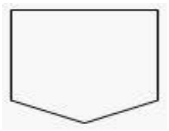


Entrada / Salida: Representa cualquier expresión de input u output

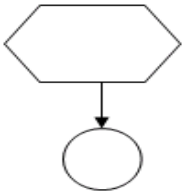


Conector de página.

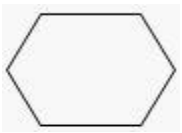
Representa impresión



Representa la conexión fuera de página.



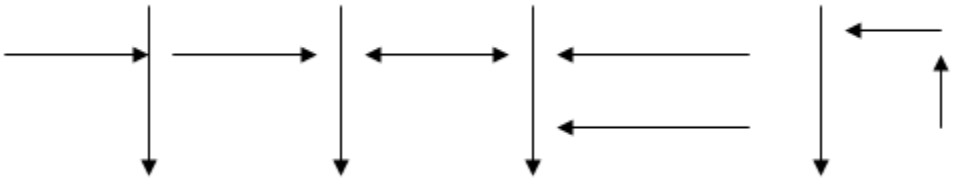
Representa operación repetitiva.



Representa proceso subalterna.

Representa datos almacenados en una cinta magnética.

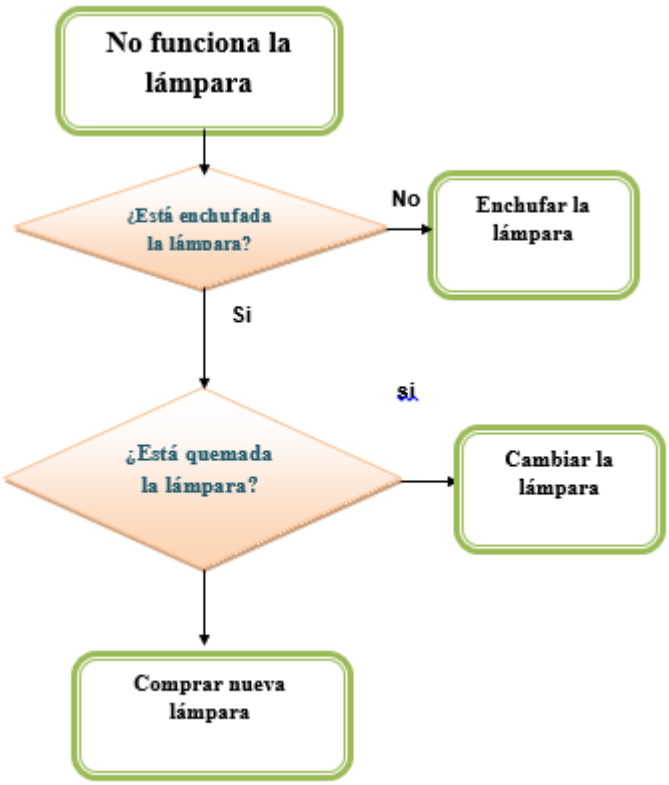
Representa datos almacenados en un Disco Magnético.



El sentido es de arriba hacia abajo en los diagramas de flujo.

→ Flecha de dirección del diagrama de flujo.

Ejemplo:



1.3 Características de los Algoritmos

Todo algoritmo debe tener como características fundamentales:

- Debe ser preciso e indicar el orden de cada paso.
- Debe estar definida. Si se procesa dos veces, el resultado debe ser igual.
- Debe ser finito.

La definición de un algoritmo debe definir tres partes: Entrada Proceso y Salida En el algoritmo de receta de cocina citado anteriormente se tendrá Entrada: ingrediente y utensilios empleados.

Proceso: elaboración de la receta en la cocma Salida: terminación del plato (por ejemplo, cordero).

Ejemplo de Algoritmo:

Un cliente ejecuta un pedido a una fábrica Esta examina en su banco de datos La ficha del cliente; si el cliente es solvente entonces la empresa acepta el pedido, en caso contrario rechazara el pedido. Redactar el algoritmo correspondiente.

Los pasos del algoritmo son:

- inicio
- leer el pedido
- Examinar la ficha del cliente
- si el cliente es solvente aceptar pedido, en caso contrario, rechazar pedido
- fin

Diseño del Algoritmo:

> En la etapa de análisis del proceso de programación se detetmma que hace el programa En la etapa de diseño se determina como hace el programa la tarea solicitada Los métodos más eficaces pera el proceso de diseño se basan en el conocido por Divide y Vencerás, es decir, la resolución de un problema complejo se realiza dividiendo el problema en sub problemas y a conmuadón dividir estos sub problemas en otros de nivel mas bajo, hasta que pueda ser naplementada una solución en la computadora. Este método se conoce técnicamente como diseño descendente (TopDown) o modular El proceso de romper el problema es cada etapa y expresar cada paso es forma más detallada se denomina refinamiento sucesivo. (Joyanes, 2013, p.34).

Cada sub programa es resuello mediante un módulo (sub programa) que tiene un solo punto de entrada y un solo punzo de salida.

> Cualquier programa bien diseñado consta de un programa principal (el módulo de nivel más alto) que llama a sub programas (módulos de nivel más bajo) que a su vez pueden llamar a otros sub programas. Los programas estructurados de esta fauna se dice que tienen un diseño modular y el método de romper el programa en módulos más pequeño se llama Programación Modular. Los módulos pueden ser planeados, codificados, comprobados y depurados independientemente (meloso por diferentes programadores) y acontinuación combinarlos entre si. (Joyanes, 2013, p.45)

El proceso implica la ejecución de los siguientes pasos hasta que el programa se termina:

- programar modulo.
- Comprobar modulo.
- Si es necesario, depurar el modulo.
- Combinar el modulo con los módulos anteriores.

El proceso que convierte los resultados del análisis del problema en un diseño modular con refinamiento sucesivo que petmitan una posterior traducción al lenguaje se denomina diseño de algoritmo.

Capítulo II: Programación

2.1 Programación

De Acuerdo a Santo M., - Patiño I., (2016):

Se llama programación a la implementación de un algoritmo en un determinado lenguaje de programación, para programar. Programar, es la acción de escribir instrucciones correctas para que sean interpretadas por una máquina.

Von Neumann en 1946 lo define como un conjunto de instrucciones que sigue la computadora para alcanzar un resultado específico.

Por lo que puedo decir que un Lenguaje de Programación, es un conjunto de reglas para comunicar ideas.

2.2 Lenguajes de programación

Un lenguaje de programación es un conjunto e instrucciones que ordenadas lógicamente y secuencialmente cumpliendo las normas de sintaxis considerada cumple como objetivo resolver un problema de procesamiento de datos.

> Cumple reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación. (Santo M., - Patiño I. 2016, p. 67)

También se define a la programación como conjunto de procedimientos que permite encontrar soluciones a través de:

- Desarrollo lógico del programa.
- Escritura del programa empleando un lenguaje de programación específico (codificación del programa).
- Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.
- Prueba y depuración del programa.
- Desarrollo de la documentación.

De acuerdo a Izquierdo L., 2015:

> Existe un error común que trata por sinónimos los términos lenguaje de programación y lenguaje informático. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo HTML (lenguaje para el marcado de páginas web que no es propiamente un lenguaje de programación, sino un conjunto de instrucciones que permiten estructurar el contenido de los documentos). Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de

> instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa. (p.56).

2.2.1 Lenguaje de Maquina

Es el que entiende las partes internas del computador, es decir, son secuencias de unos y ceros que forman las instrucciones que entienden directamente el procesador. Lógicamente, los lenguajes maquina son dependientes del hardware ya que cada procesador posee un conjunto de instrucciones diferentes.

Las ventajas son:

- No necesita traductores del lenguaje

Las desventajas son:

- Dificultad en la escritura y comprensión de los programas.
- Poca fiabilidad, ya que es fácil cometer errores de escritura.
- Costo alto, ya que la escritura de programa es lenta.
- Programas totalmente dependientes del hardware.

Por todo ello, este tipo de lenguajes no se utiliza para programar. Sin embargo hay que tener en cuenta todos los programas para poder ser ejecutados, en algún momento, deberán traducirse al lenguaje máquina que es el único lenguaje que entiende la computadora.

Se entiende que escribir en el lenguaje maquina 0 y 1 es incómodo y muy complicado. Históricamente a la hora de diseñar un algoritmo para que el computador

ejecutara, se escribía mediante unas etiquetas nemotécnicas; este fue el origen del lenguaje ensamblador.

Por ejemplo:

Quizás en una cierta arquitectura la instrucción de borrado de memoria es decir **memoryclear** en ingles corresponde al código 010.

Pronto surgieron programas que leían, siguiendo el ejemplo, MC, y lo sustituían por 010.

2.2.2 Lenguaje Ensamblador

El código máquina tenía inconvenientes para programar:

- Son instrucciones difíciles de recordar ya que no guardan relación con la operación que se está realizando.
- Hay diferencias entre las instrucciones de un procesador a otro.

Esto motivo a darles nombre a las instrucciones con un nombre sencillo que trate de reconocer la operación. Esto resulta una mayor comodidad a los programadores y se hace más sencillo programar con comandos ya definidos.

Con estos antecedentes se fueron construyendo lenguajes de programación cada vez más sencillos en su manejo. Estos son llamados los códigos fuente; siguen las reglas sintácticas de un determinado lenguaje de programación. Existen numerosos lenguajes de programación, y se utiliza uno u otros según sus características se adecúen más o menos a la resolución de nuestro problema.

2.3 Tipos de Lenguajes de Programación

De acuerdo al nivel de abstracción, como se ejecuta y según su paradigma de programación, pueden ser:

Según su nivel de abstracción:

- Lenguajes de bajo nivel
- Lenguajes de medio nivel
- Lenguajes de alto nivel

Según la forma de ejecución:

- Lenguajes compilados
- Lenguajes interpretados

Según el paradigma de programación:

- Lenguajes imperativos
- Lenguajes funcionales
- Lenguajes lógicos
- Lenguajes orientados a objetos

2.3.1 Lenguajes de Alto Nivel

Comienzan utilizando comandos generalmente verbos en inglés, se produce, por tanto, una abstracción de datos, muy deseable para poder utilizar el trabajo de otros para avanzar un paso más en vez de tener que reinventar la rueda, como se suele decir. Estos textos en los que se codifican los algoritmos son los códigos fuente; siguen las reglas sintácticas de un determinado lenguaje de programación. Existen

> numerosos lenguajes de programación, y se utiliza uno u otros según sus características se adecúen más o menos a la resolución de nuestro problema. Son BASIC, C, C++, Java, etc.. (Joyanes, 2015, p.86).

2.3.2 Traductores de lenguaje

Lenguajes compilados

Naturalmente, un programa que se escribe en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente. Se requiere una corrida de compilación antes de procesar los datos de un problema (Cibertec, 2008).

Los compiladores son aquellos cuya función es traducir un programa escrito en un determinado lenguaje a un idioma que la computadora entienda (lenguaje máquina con código binario).

Al usar un lenguaje compilado (como lo son los lenguajes del popular Visual Studio de Microsoft), el programa desarrollado nunca se ejecuta mientras haya errores, sino hasta que luego de haber compilado el programa, ya no aparecen errores en el código

Lenguajes interpretados

Se puede también utilizar una alternativa diferente de los compiladores para traducir lenguajes de alto nivel. En vez de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la corrida de compilación para utilizarlo en una corrida de producción futura, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un

programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el proceso de los datos. No se graba el código objeto para utilizarlo posteriormente (Cibertec, 2008).

> La siguiente vez que se utilice una instrucción, se le debe interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo, cada instrucción del ciclo tendrá que volver a ser interpretado cada vez que se ejecute el ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una corrida de compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una corrida de producción por ejemplo 20578946426312015 84. (Cibertec, 2008, p.36)

2.4 Metodologías de Programación

La evolución de los lenguajes de programación ha ido paralela a la idea de metodologías de programación: enfoques alternativos a los procesos de programación. En realidad una metodología de programación representa fundamentalmente enfoques diferentes para la construcción de soluciones a problemas y por consiguiente afectan al proceso completo de desarrollo de software. Las metodologías de programación clásicos son: procedimental (estructurada y modular), funcional, declarativo y orientado a objetos.

2.4.1 Programación Estructurada y Modular

Programación Estructurada es una técnica en la cual la estructura de un programa, esto es, la interpelación de sus partes realiza tan claramente cómo es posible mediante el uso de tres estructuras lógicas de control:

Secuencia: Sucesión simple de dos o más operaciones.

Selección: bifurcación condicional de una o más operaciones.

Iteración: Repetición de una operación mientras se cumple una condición.

Estos tres tipos de estructuras lógicas de control pueden ser combinados para producir programas que manejen cualquier tarea de procesamiento de información.

Un programa estructurado está compuesto de segmentos, los cuales puedan estar constituidos por unas pocas instrucciones o por una página o más de codificación. Cada segmento tiene solamente una entrada y una salida, estos segmentos, asumiendo que no poseen lazos infinitos y no tienen instrucciones que jamás se ejecuten, se denominan programas propios. Cuando varios programas propios se combinan utilizando las tres estructuras básicas de control mencionadas anteriormente, el resultado es también un programa propio (Patiño, 2016).

> La programación Estructurada está basada en el Teorema de la Estructura, el cual establece que cualquier programa propio (un programa con una entrada y una salida exclusivamente) es equivalente a un programa que contiene solamente las estructuras lógicas mencionadas anteriormente (Patiño, 2016, p.45).

Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación. Esto es importante debido a que, es mucho más fácil comprender completamente el trabajo que realiza una función determinada, si todas las instrucciones que influyen en su acción están físicamente contiguas y encerradas por un bloque. La facilidad de lectura, de comienzo a fin, es una consecuencia de utilizar solamente tres estructuras de control y de eliminar la instrucción de desvío de flujo de control, excepto en circunstancias muy especiales tales como la simulación de una estructura lógica de control en un lenguaje de programación que no la posea (Santo, 2016).

Ventajas potenciales

Un programa escrito de acuerdo a estos principios no solamente tendrá una estructura, sino también una excelente presentación.

Un programa escrito de esta forma tiende a ser mucho más fácil de comprender que programas escritos en otros estilos.

La facilidad de comprensión del contenido de un programa puede facilitar el chequeo de la codificación y reducir el tiempo de prueba y depuración de programas. Esto último es cierto parcialmente, debido a que la programación estructurada concentra los errores en uno de los factores más generador de fallas en programación: la lógica.

Un programa que es fácil para leer y el cual está compuesto de segmentos bien definidos tiende a ser simple, rápido y menos expuesto a mantenimiento. Estos beneficios derivan

en parte del hecho que, aunque el programa tenga una extensión significativa, en documentación tiende siempre a estar al día, esto no suele suceder con los métodos convencionales de programación.

Santo (2016) “La programación estructurada ofrece estos beneficios, pero no se la debe considerar como una panacea ya que el desarrollo de programas es, principalmente, una tarea de dedicación, esfuerzo y creatividad.”(p.45)

Programación Modular

> La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable. Se presenta históricamente como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos de lo que ésta puede resolver (Patiño, 2016, p.74).

Al aplicar la programación modular, un problema complejo debe ser dividido en varios sub problemas más simples, y estos a su vez en otros sub problemas más simples. Esto debe hacerse hasta obtener sub problemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación. Ésta técnica se llama refinamiento sucesivo, divide y vencerás ó análisis descendente (Top-Down).

2.4.2 Programación Orientadas a Objetos

La programación orientada a objetos (POO, u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos en sus interacciones, para diseñar aplicaciones y programas informáticos.

Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

> Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos. Un lenguaje orientado a objetos es un lenguaje de programación que permite el diseño de aplicaciones orientadas a objetos. Dicho esto, Luis Izquierdo nos dice lo normal es que toda persona que vaya a desarrollar aplicaciones orientadas a objetos aprenda primero la filosofía es decir que adquiera la forma de pensar y después el lenguaje, porque filosofía sólo hay una y lenguajes muchos (Patiño, 2016, p.75).

Los lenguajes de programación tradicionales no orientados a objetos, como C, Pascal, BASIC, basan su funcionamiento en el concepto de procedimiento o función.

Una función es simplemente un conjunto de instrucciones que operan sobre unos argumentos y producen un resultado. De este modo, un programa no es más que una sucesión de llamadas a funciones, ya sean éstas del sistema operativo, proporcionadas por el propio lenguaje, o desarrolladas por el mismo usuario.

Si nos detenemos a pensar sobre cómo se nos plantea un problema cualquiera en la realidad podremos ver que lo que hay en la realidad son agentes u objetos. Estas entidades poseen un conjunto de propiedades o atributos, y un conjunto de métodos mediante los cuales muestran su comportamiento. Y no sólo eso, también podremos descubrir, a poco que nos fijemos, todo un conjunto de interrelaciones entre las entidades, guiadas por el intercambio de mensajes; las entidades del problema responden a estos mensajes mediante la ejecución de ciertas acciones. El siguiente ejemplo, nos mostrara lo dicho anteriormente.

Si nos imaginemos la siguiente situación:

Un domingo por la tarde estoy en casa viendo la televisión, y de repente mi madre siente un fuerte dolor de cabeza; como es natural, lo primero que hago es tratar de encontrar una caja de aspirinas. Lo que acabo de describir es una situación que probablemente no resulte muy extraña a muchos de nosotros (Patiño, 2016).

Vamos a verla en clave de objetos: el objeto hijo ha recibido un mensaje procedente del objeto madre. El objeto hijo responde al mensaje o evento ocurrido mediante una acción: buscar aspirinas. La madre no tiene que decirle al hijo dónde debe buscar, es responsabilidad del hijo resolver el problema como considere más oportuno. Al objeto madre le basta con haber emitido un mensaje.

Continuemos con la historia. El hijo no encuentra aspirinas en el botiquín y decide acudir a la farmacia de guardia más cercana para comprar aspirinas. En la farmacia es atendido por una señorita que le pregunta qué desea, a lo que el hijo responde: una caja de aspirinas, por favor.

La farmacéutica desaparece para regresar al poco tiempo con una caja de aspirinas en la mano. El hijo paga el importe, se despide y vuelve a su casa. Allí le da un comprimido a su madre, la cual al cabo de un rato comienza a experimentar una notable mejoría hasta la completa desaparición del dolor de cabeza. El hijo, como objeto responsable de un cometido, sabe lo que debe hacer hasta conseguir una aspirina. Para ello entra en relación con un nuevo objeto, la farmacéutica, quien responde al mensaje o evento de petición del objeto hijo con la búsqueda de la aspirina. El objeto farmacéutica

es ahora el responsable de la búsqueda de la aspirina. El objeto farmacéutica lanza un mensaje al objeto hijo solicitando el pago del importe, y el objeto hijo responde a tal evento con la acción de pagar (Patiño, 2016).

Se observa, en esta situación objetos que se diferenciaban de los demás por un conjunto de características o propiedades, y por un conjunto de acciones que realizaban en respuesta a unos eventos que se originaban en otros objetos o en el entorno.

También podemos darnos cuenta de que, aunque todos los objetos tienen propiedades distintas, como el color del cabello, el grado de simpatía o el peso, todos tienen un conjunto de atributos en común por ser ejemplos de una entidad superior llamada ser humano.

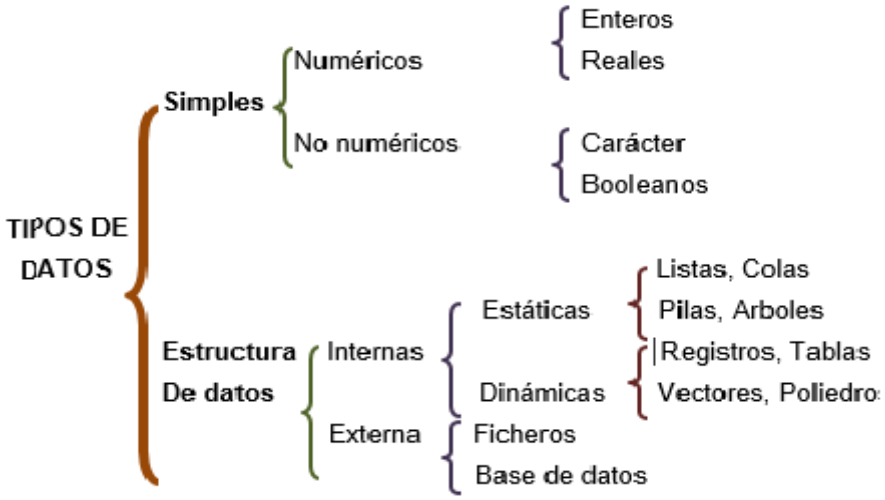
2.5 Estructura de Datos

En ciencias de la computación, una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

Las estructuras de datos son un medio para manejar grandes cantidades de datos de manera eficiente para usos tales como grandes bases de datos y servicios de indización de Internet. Por lo general, las estructuras de datos eficientes son clave para diseñar algoritmos eficientes. Algunos métodos formales de diseño y lenguajes de programación destacan las estructuras de datos, en lugar de los algoritmos, como el factor clave de organización en el diseño de software.

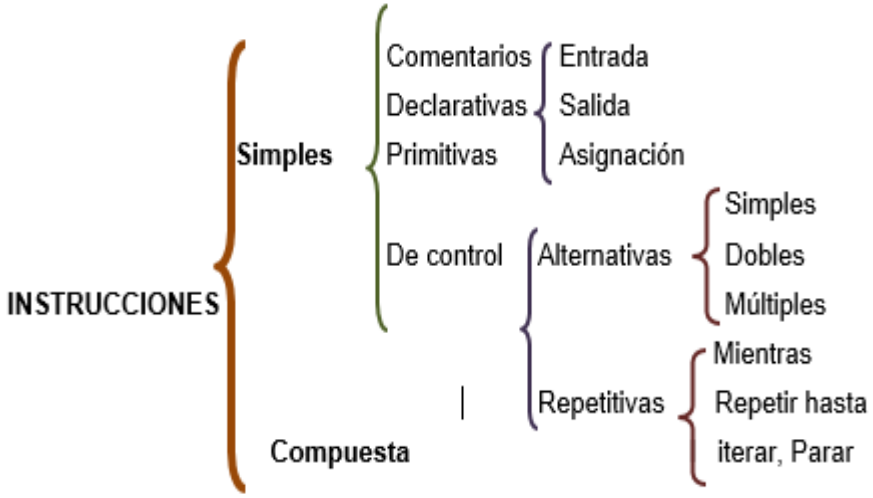
2.5.1 Tipos de datos

Un programa será capaz de manejar muchos tipos de dato, tanto simples como: numéricos, alfanuméricos, booleanos, también estructuras más complejas como: tablas, ficheros,… los datos que pueden manejar un programa se puede clasificar en:



2.5.2 Clasificación de instrucciones

Las instrucciones se pueden clasificar como sigue a continuación:



2.5.3 Variables y constantes

Según Tapara (2010) nos redacta puntualmente que:

> Como hemos visto, el computador sigue una serie de instrucciones. Pero esas instrucciones tienen que operar sobre una serie de datos. El computador típico sólo procesa una instrucción a la vez, por lo que necesita espacios de memoria donde guardar o depositar, a modo de cajones, por usar un símil conocido, los diversos datos con los que trabaja. Aquí es donde entran en juego las variables y constantes.
>
> En los inicios, con el ensamblador, se podía decir al computador, por ejemplo: Ejecuta la instrucción de esa posición de memoria o también En esa posición de memoria está guardada mi edad, imprímela por pantalla. Todo esto se deriva del hecho de que los programas también son datos. Esta ambigüedad presenta numerosos inconvenientes cuando se producen errores, como el lector se imaginará fácilmente: de ahí que, a medida que los lenguajes promocionan hacia niveles superiores, se impida el tratamiento indistinto de los datos. A partir de entonces, un programa tiene que decirle al sistema operativo los cajones que necesita y éste se los proporciona independientemente de cuáles sean (p.45).

Quizás suene más complicado de lo que es. Un ejemplo: Queremos sumar dos números. Nuestro programa tendrá que tener tres cajones: Uno para cada número y otro para el resultado. Cada cajón tiene un nombre en vez de una posición de memoria, de manera que sólo hay que nombrarlo:

Necesito cajones A, B y Resultado

Lee un número y guárdalo en A

Lee un número y guárdalo en B

Suma A y B y guárdalo en Resultado

Imprime el contenido de Resultado

He aquí nuestro programa. Como cabe pensar, un procesador no tiene la instrucción **Imprime por pantalla**; esto es una llamada a otra porción de código que, gracias a la abstracción, nosotros o no hemos escrito, o hemos escrito una sola vez; a partir de lo cual podemos imprimir todo el texto que queramos en la pantalla.

Las posiciones de memoria A y B son Variables. Si queremos leerlas o escribirlas, podemos hacerlo. Típicamente, existirán datos que no pensamos modificar; no querremos que el usuario tenga que introducirlos cada vez, pues son de naturaleza más constante que otros (como puede ser el valor Pi para calcular el perímetro o área de un círculo). Para evitar modificarlos por error, podemos pedir al sistema variables especiales, que no puedan ser reescritas. Son las Constantes. Un ejemplo:

```
Comentario: Este programa calcula el área de un círculo
Constante PI = 3,14159265
Variable R
Variable Resultado
Leer número y guardar en R
Calcular PI * (R * R) y guardar en Resultado
Imprimir Resultado
```

El uso de variables y constantes se asemeja al uso que se les da en el álgebra o en otras ramas matemáticas.

Nótese también la clara separación entre estructuras de datos y algoritmos. Según los lenguajes, esto puede ser o no obligatorio, pero es recomendable en aras de una mayor claridad del trabajo.

2.5.4 Procedimientos y funciones

Procedimientos

Cuando un programa comienza a ser largo y complejo (la mayoría de los problemas reales se solucionan con programas de este tipo) no es apropiado tener un único texto con sentencias una tras otra. La razón es que no se comprende bien qué hace el programa debido a que se intenta abarcar toda la solución a la vez. Asimismo el programa se vuelve monolítico y difícil de modificar. Además suelen aparecer trozos de código muy similares entre sí repetidos a lo largo de todo el programa (Cairo, 2013).

Para solucionar estos problemas y proporcionar otras ventajas adicionales a la programación, los lenguajes de alto nivel suelen disponer de una herramienta que permite estructurar el programa principal como compuesto de subprogramas (rutinas) que resuelven problemas parciales del problema principal. A su vez, cada uno de estos subprogramas puede estar resuelto por otra conjunción de problemas parciales, etc.

Los procedimientos y las funciones son mecanismos de estructuración que permiten ocultar los detalles de la solución de un problema y resolver una parte de dicho problema en otro lugar del código (Cairo, 2013).

En su concepción más simple, un procedimiento es una construcción que permite dar nombre a un conjunto de sentencias y declaraciones asociadas que se usan para

resolver un subproblema dado. Usando procedimientos la solución es más corta, comprensible y fácilmente modificable.

Funciones

Una función es un objeto del ambiente, con nombre, tipo y valor único. El tipo se asocia al valor que retorna la función cuando es evaluada para un conjunto dado de valores de sus argumentos.

Función nombre (lista de parámetros formales): Tipo de resultado

    Declaración de variables

Inicio

    Acciones

    Devolver (constante, variable o expresión)

Fin función

Capítulo III

Lenguaje de programación C++ y C SHARD

3.1. Lenguaje de programación C++

Comenzaremos estudiando el soporte del C++ a la programación imperativa, es decir, la forma de definir y utilizar los tipos de datos, las variables, las operaciones aritméticas, las estructuras de control y las funciones. Es interesante remarcar que toda esta parte está heredada del C, por lo que también sirve de introducción a este lenguaje.
El mínimo programa de C++ es: main() { }

Lo único que hemos hecho es definir una función (main) que no tiene argumentos y no hace nada. Las llaves { } delimitan un bloque en C++, en este caso el cuerpo de la función main.

Todos los programas deben tener una función main() que es la que se ejecuta al comenzar el programa.

Un programa será una secuencia de líneas que contendrán sentencias, directivas de compilación y comentarios. Las sentencias simples se separan por punto y coma y las compuestas se agrupan en bloques mediante llaves.

Las directivas serán instrucciones que le daremos al compilador para indicarle que realice alguna operación antes de compilar nuestro programa, las directivas comienzan con el símbolo # y no llevan punto y coma.

Los comentarios se introducirán en el programa separados por /* y */ o comenzándolos con //. Los comentarios entre /* y */ pueden tener la longitud que queramos, pero no se anidan, es decir, si escribimos /* hola /* amigo */ mío */, el compilador interpretará que el comentario termina antes de mío, y dará un error. Los comentarios que comienzan por // sólo son válidos hasta el final de la línea en la que aparecen.

Un programa simple que muestra todo lo que hemos visto puede ser el siguiente:

```
/*
    Este es un programa mínimo en C++, lo único que hace es escribir una frase en la
    pantalla
*/
#include <iostream.h>
int main ()
{
 cout << "Hola amigo"; // imprime en la pantalla la frase "hola amigo"
}
```

La primera parte separada entre /* y */ es un comentario. Es recomendable que se comenten los programas, explicando que es lo que estamos haciendo en cada caso, para que cuando se lean sean más comprensibles.

La línea que empieza por # es una directiva. En este caso indica que se incluya el fichero "iostream.h", que contiene las definiciones para entrada/salida de datos en C++. En la declaración de main () hemos incluido la palabra int, que indica que la función devuelve un entero. Este valor se le entrega al sistema operativo al terminar el programa. Si no se devuelve ningún valor el sistema recibe un valor aleatorio.

La sentencia separada ente llaves indica que se escriba la frase "Hola guapo". El operador << ("poner en") escribe el segundo argumento en el primero. En este caso la cadena "Hola guapo\n" se escribe en la salida estándar (cout). El carácter \ seguido de otro carácter indica un solo carácter especial, en este caso el salto de línea (\n).

Veremos el tema de la entrada salida estándar más adelante. Hay que indicar que las operaciones de E/S se gestionan de forma diferente en C y C++, mientras que el C proporciona una serie de funciones (declaradas en el fichero "stdio.h"), el C++ utiliza el concepto de stream, que se refiere al flujo de la información (tenemos un flujo de entrada que proviene de cin y uno de salida que se dirige a cout) que se maneja mediante operadores de E/S.

Por último señalar que debemos seguir ciertas reglas al nombrar tipos de datos, variables, funciones, etc. Los identificadores válidos del C++ son los formados a partir de los caracteres del alfabeto (el inglés, no podemos usar ni la ñ ni palabras acentuadas),

los dígitos (0...9) y el subrayado (_), la única restricción es que no podemos comenzar un identificador con un dígito (es así porque se podrían confundir con literales numéricos). Hay que señalar que el C++ distingue entre mayúsculas y minúsculas, por lo que Hola y hola representan dos cosas diferentes. Hay que evitar el uso de identificadores que sólo difieran en letras mayúsculas y minúsculas, porque inducen a error.

3.2 Tipos de datos y operadores

Los tipos elementales definidos en C++ son:

char, short, int, long, que representan enteros de distintos tamaños (los caracteres son enteros de 8 bits)

float, double y long double, que representan números reales (en coma flotante).

Para declarar variables de un tipo determinado escribimos el nombre del tipo seguido del de la variable. Por ejemplo:

```
int i;
double d;
char c;
```

Sobre los tipos elementales se pueden emplear los siguientes operadores aritméticos:

+ (Más, como signo o como operación suma)

- (menos, como signo o como operación resta)

* (Multiplicación)

/ (División)

% (resto)

Y los siguientes operadores relacionales:

== (igual)

!= (distinto)

< (menor que)

> (mayor que)

<= (menor o igual que)

>= (mayor o igual que)

El operador de asignación se representa por =.

En la bibliografía del C++ se suelen considerar como tipos derivados los construidos mediante la aplicación de un operador a un tipo elemental o compuesto en su declaración. Estos operadores son:

* Puntero

& Referencia

[] Vector (Array)

() Función

Los tipos compuestos son las estructuras (struct), las uniones (unión) y las clases (class).

Estructuras de control

Como estructuras de control el C++ incluye las siguientes construcciones:

Condicionales:

if instrucción de selección simple

switch instrucción de selección múltiple

Bucles:

do-while instrucción de iteración con condición final

while instrucción de iteración con condición inicial

for instrucción de iteración especial (similar a las de repetición con contador)

De salto:

break	instrucción de ruptura de secuencia (sale del bloque de un bucle o instrucción condicional)
continue	instrucción de salto a la siguiente iteración (se emplea en bucles para saltar a la posición donde se comprueban las condiciones)
goto	instrucción de salto incondicional (salta a una etiqueta)
return	instrucción de retorno de un valor (se emplea en las funciones)

3.3 Programación eficiente

Veremos una serie de mecanismos del C++ útiles para hacer que nuestros programas sean más eficientes. Comenzaremos viendo cómo se organizan y compilan los programas, y luego veremos que construcciones nos permiten optimizar los programas.

Estructura de los programas

Casariego (2010) hace mención que:

> El código de los programas se almacena en ficheros, pero el papel de los ficheros no se limita al de mero almacén, también tienen un papel en el lenguaje: son un ámbito para determinadas funciones (estáticas y en línea) y variables (estáticas y constantes) siempre que se declaren en el fichero fuera de una función.
>
> Además de definir un ámbito, los ficheros nos permiten la compilación independiente de los archivos del programa, aunque para ello es necesario proporcionar declaraciones con la información necesaria para analizar el archivo de forma aislada (p.74).

Una vez compilados los distintos ficheros fuente (que son los que terminan en .c, .cpp, etc.), es el linker el que se encarga de enlazarlos para generar un sólo fichero fuente ejecutable.

En general, los nombres que no son locales a funciones o a clases se deben referir al mismo tipo, valor, función u objeto en cada parte de un programa.

Si en un fichero queremos declarar una variable que está definida en otro fichero podemos hacerlo declarándola en nuestro fichero precedida de la palabra extern.
Si queremos que una variable o función sólo pertenezca a nuestro fichero la declaramos static.

Si declaramos funciones o variables con los mismos nombres en distintos ficheros producimos un error (para las funciones el error sólo se produce cuando la declaración es igual, incluyendo los tipos de los parámetros).

Las funciones y variables cuyo ámbito es el fichero tienen enlazado interno (es decir, el linker no las tiene en cuenta).

Los ficheros cabecera

Una forma fácil y cómoda de que todas las declaraciones de un objeto sean consistentes es emplear los denominados ficheros cabecera, que contienen código ejecutable y/o definiciones de datos. Estas definiciones o código se corresponderán con la parte que queremos utilizar en distintos archivos.

Para incluir la información de estos ficheros en nuestro fichero .c empleamos la directiva include, que le servirá al preprocesador para leer el fichero cabecera cuando compile nuestro código.

Un fichero cabecera debe contener:

Definición de tipos	struct punto { int x, y; };
Templates	template <class T> class V { … }
Declaración de funciones	extern int strlen (const char *);
Definición de funciones inline	inline char get { return *p++ ;}
Declaración de variables	extern int a;
Definiciones constantes	const float pi = 3.141593;
Enumeraciones	enum bool { false, true };
Declaración de nombres	class Matriz;
Directivas include	#include <iostream.h>
Definición de macros	#define Case break;case
Comentarios	/* cabecera de mi_prog.c */

Y no debe contener:

Definición de funciones ordinarias	char get () { return *p++}
Definición de variables	int a;
Definición de agregados constantes	const tabla[] = { … }

Si nuestro programa es corto, lo más usual es crear un solo fichero cabecera que contenga los tipos que necesitan los diferentes ficheros para comunicarse y poner en

estos ficheros sólo las funciones y definiciones de datos que necesiten e incluir la cabecera global.

Si el programa es largo o usamos ficheros que pueden ser reutilizados lo más lógico es crear varios ficheros cabecera e incluirlos cuando sea necesarios.

Por último indicaremos que las funciones de biblioteca suelen estar declaradas en ficheros cabecera que incluimos en nuestro programa para que luego el linker las enlace con nuestro programa. Las bibliotecas estándar son:

Bibliotecas de C:

assert.h	Define la macro assert()
ctype.h	Manejo de caracteres
errno.h	Tratamiento de errores
float.h	Define valores en coma flotante dependientes de la implementación
limits.h	Define los límites de los tipos dependientes de la implementación
locale.h	Define la función setlocale()
math.h	Definiciones y funciones matemáticas
setjmp.h	Permite saltos no locales
signal.h	Manejo de señales
stdarg.h	Manejo de listas de argumentos de longitud variable
stddef.h	Algunas constantes de uso común
stdio.h	Soporte de E/S
sdlib.h	Algunas declaraciones estándar
string.h	Funciones de manipulación de cadenas
time.h	Funciones de tiempo del sistema

Bibliotecas de C++:

fstream.h	Streams fichero
iostream.h	Soporte de E/S orientada a objetos (streams)
new.h	Definición de _new_handler
strstream.h	Definición de streams cadena

El preprocesador

El preprocesador es un programa que se aplica a los ficheros fuente del C++ antes de compilarlos. Realiza diversas tareas, algunas de las cuales podemos controlarlas nosotros mediante el uso de directivas de preprocesado. Como veremos, estas directivas nos permiten definir macros como las de los lenguajes ensambladores (en realidad no se trata más que de una sustitución).

A continuación veremos las fases de preprocesado y las directivas, así como una serie de macros predefinidas. Por último explicaremos lo que son las secuencias trigrafo.

Fases de preprocesado

1. Traduce los caracteres de fin de línea del fichero fuente a un formato que reconozca el compilador. Convierte los trigrafos en caracteres simples.
2. Concatena cada línea terminada con la barra invertida (\) con la siguiente.
3. Elimina los comentarios. Divide cada línea lógica en símbolos de preprocesado y espacios en blanco.
4. Ejecuta las directivas de preprocesado y expande los macros.
5. Reemplaza las secuencias de escape dentro de constantes de caracteres y cadenas de literales por sus caracteres individuales equivalentes.
6. Concatena cadenas de literales adyacentes.

7. Convierte los símbolos de preprocesado en símbolos de C++ para formar una unidad de compilación.

Estas fases se ejecutan exactamente en este orden.

Directivas del preprocesador

#define ID VAL	Define la macro ID con valor VAL
#include "fichero"	Incluye un fichero del directorio actual
#include <fichero>	Incluye un fichero del directorio por defecto
#defined id	Devuelve 1 si id está definido
#defined (id)	Lo mismo que el anterior
#if expr	Si la expresión se cumple se compila todo lo que sigue. Si no se pasa hasta un #else o un #endif
#ifdef id	Si el macro id ha sido definido con un #define la condición se cumple y ocurre lo del caso anterior. Es equivalente a if defined id
#ifndef id	Si el macro id no ha sido definido con un #define, la condición se cumple. es equivalente a if !defined id
#else	Si el #if, #ifdef o #ifndef más reciente se ha cumplido todo lo que haya después del #else hasta #endif no se compila. Si no se ha cumplido si se compila
#elif expr	Contracción de #else if expr
#endif	Termina una condición

#line CONST ID Cambia el número de línea según la constante CONST y el nombre del fichero de salida de errorea a ID. Modifica el valor de los macros predefinidos __LINE__ y __FILE__

#pragma OPCION Especifica al compilador opciones específicas de la implementación

#error CADENA Causa la generación de un mensaje de error con la cadena dada i = MIN(j*3, k-1);

}

Después del preprocesado tendremos:

```
main () {
        int i, j=6, k=8;
        i = (((j*3) < (k-1)) ? (j*3) : (k-1));
}
```

Si no hubiéramos puesto paréntesis en la definición de la macro, al sustituir a y b podíamos haber introducido operadores con mayor precedencia que?: y haber obtenido un resultado erróneo al ejecutar la macro. Notar que la macro no hace ninguna comprobación en los parámetros simplemente sustituye, por lo que a veces puede producir resultados erróneos.

3.4 Aplicaciones con C++

Programa que calcule la suma de dos números.

```
#include <stdio.h>
void main(){
    int X,Y,Z;
    printf("Por favor, introduzca un numero:  ");
    scanf("%d",&X);
    printf("Ahora, inserte otro:  ");
    scanf("%d",&Y);
    Z=X+Y;
    printf("\nEl resultado es %d\n",Z);
}
```

Programa que calcula la media aritmética de tres números cualesquiera.

```
#include <stdio.h>
void main(){
	float a,b,c,M;
	printf("Bienvenido, calcularemos la media aritmetica de tres numeros.\n\n");
	printf("Por favor, introduzca el primero: ");
	scanf("%f",&a);
	printf("Ahora, inserte el segundo de ellos: ");
	scanf("%f",&b);
	printf("Por ultimo, teclee el numero final: ");
	scanf("%f",&c);
	M=(a+b+c)/3;
	printf("\nEl resultado es %f\n\n",M);
}
```

Programa que calcula el área de un triángulo (Fórmula de Herón).

```
#include <stdio.h>
#include <math.h>
void main(){
    float a,b,c,sp,R;
    printf("Bienvenido. Calcularemos el area del triangulo.\n\n");
    printf("Introduce el primer lado: ");
    scanf("%f",&a);
    printf("Ahora, inserta el segundo lado: ");
    scanf("%f",&b);
    printf("Por último, escribe el tercer lado: ");
    scanf("%f",&c);
    sp=(a+b+c)/2;
    R=sqrt(sp*(sp-a)*(sp-b)*(sp-c));
    printf("\nEl area obtenida es %f\n\n",R);
    printf("Muchas gracias por utilizar este progama.\n\n");
    }
```

3.5 C SHARD

C# o C Sharp es un lenguaje de programación que está incluido en la Plataforma .NET y corre en el Lenguaje Común en Tiempo de Ejecución (CLR, Common Language Runtime). El primer lenguaje en importancia para el CLR es C#, mucho de lo que soporta la Plataforma .NET está escrito en C#.

C# deriva de C y C++, es moderno, simple y enteramente orientado a objetos, simplifica y moderniza a C++ en las áreas de clases, namespaces, sobrecarga de métodos y manejo de excepciones. Se eliminó la complejidad de C++ para hacerlo más fácil de utilizar y menos propenso a errores.

C# es "case sensitive", es decir, que distingue mayúsculas de minúsculas. HolaMundo es diferente a holamundo.

APLICACIÓN DIDACTICA

UNIVERSIDAD NACIONAL DE EDUCACIÓN "ENRIQUE GUZMÁN Y VALLE"

Alma Máter del Magisterio Nacional

SESIÓN DE APRENDIZAJE

Título: "Aprendemos La Estructura de control simple if"

I. DATOS INFORMATIVOS

1.1	INSTITUCIÓN EDUCATIVA	: "Víctor E Vivar"
1.2	ÁREA CURRICULAR	: Educación para el trabajo
1.3	COMPONENTE	: Informática
1.4	GRADO Y SECCIÓN	: 5to
1.5	TIEMPO	: 90'
1.6	PROFESOR	: Roger Ostos Acevedo

II TEMA TRANSVERSAL: Formación ética y de talentos

III ESPECTATIVA DE LOGRO: Analiza procedimientos para aplicar una estructura de control simple **if**

IV ORGANIZACIÓN DE LOS APRENDIZAJES

CONCEPTOS	APRENDIZAJES ESPERADOS	VALOR/ACTITUDES
1. PREVIOS - Lenguaje de Programación C++ - Estructura de un programa en C++ 2. NUEVO - Estructura de control simple **if**	- Identifica una estructura de control - Analiza una estructura de control simple - Aplica la estructura de control simple en los ejercicios.	**SOLIDARIDAD** - Comparte sus conocimientos y con sus compañeros. **RESPONSABILIDAD** - Cumple con sus tareas y realiza sus trabajos con seriedad.

V SECUENCIA DIDÁCTICA

SITUACIÓN DE APRENDIZAJE	ESTRATEGIAS DIDÁCTICAS		EVALUACIÓN			T
	Métodos, procedimientos y técnicas	Medios y materiales	Criterios	Indicadores	Instrumentos	
Organización	El profesor da las recomendaciones necesarias para iniciar el trabajo en el laboratorio. Se le repartirá su ficha de trabajo.	- Cuaderno	GP	Respeta las normas de convivencia.	Ficha de Observación actitudinal	5’
Recuperación de saberse previos	Se recogerán los saberes previos en el transcurso de la clase como: ¿Qué es una estructura de control? ¿Cómo era la estructura de un programa en C++?					10’
Conflicto cognitivo	¿Cómo le decimos a la computadora que imprima una acción verdadera? Por ejemplo: Si deseamos que al introducir una ecuación de segundo grado verifique que es correcta.			Identifica la estructura de control simple **if**	Ficha de evaluación grupal	10’
Fase de acción	Los alumnos trabajan los ítems de su ficha de trabajo utilizando el programa para verificar su respuesta.	-Programa en lenguaje C++ -Guía de laboratorio	EP			10’
Fase de formulación	Los alumnos trabajan en grupo (de dos) con el fin de encontrar posibles soluciones de los ejercicios planteados.			Interpreta los códigos que se encuentran en la	Ficha de evaluación grupal	20’
Fase de validación						

(Argumentación y negociación) **Institucionalización** (saber matemático) **Transferencia** (Aplicación de saberes) **Extensión**	Los alumnos explican la solución de sus ejercicios en la pizarra y responden las preguntas. De acuerdo al trabajo de todos los grupos se procede a aclarar las dudas en el transcurso de la clase y se explica la estructura simple de una manera más formal. Se resolverán un ejercicio en la pizarra y lo comprobaran en el programa. Tarea: formular un problema, que tenga la estructura de control if, resolverlo siguiendo los 4 pasos tratados.	- Cuaderno -Programa en lenguaje C++ -Guía de laboratorio	CAT	estructura de control simple Resuelve ejercicios aplicando la estructura de control simple **if**	Ficha de evaluación grupal. Registro	15’ 5’ 5’

GUIA DE LABORATORIO:

Apellido y Nombre: …….

Profesor:

Fecha: ……/……. /…….. Grado: ……..

Escucha atentamente las indicaciones, luego responde y completa según te indique tu profesora.

1) ¿Qué es una estructura de control?

……

……

……

……

2) ¿Cómo le digo a la computadora que imprima una acción verdadera?

……

……

……

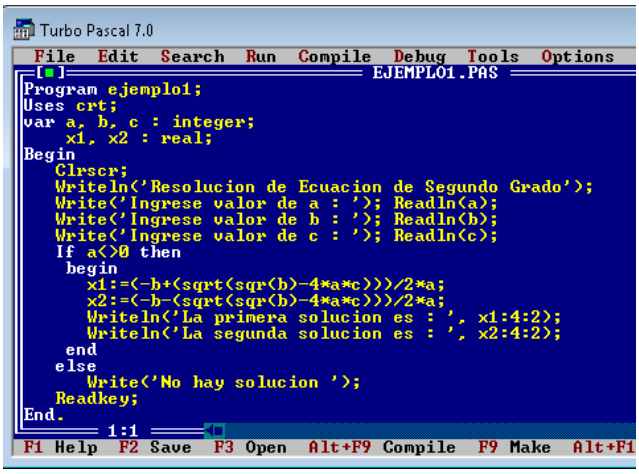
……

Por ejemplo: **Un estudiante introduce las constantes de una ecuación de segundo grado para obtener el conjunto solución.**

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

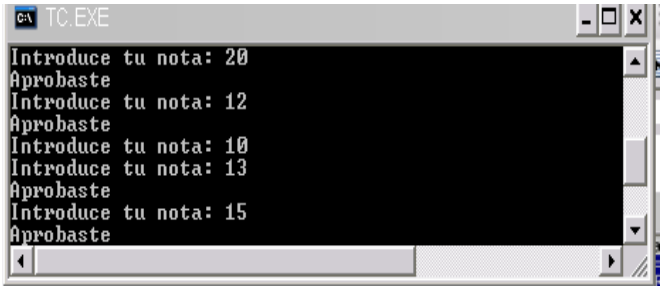
1) Pseudocódigo

3) Digitamos siguiendo la estructura del Turbo Pascal



2) Diagrama de flujo.

4) Corremos el programa con CTRL + F5 y si hay algún error nos dirá y no mostrara ningún resultado.



3) Resuelve los siguientes problemas (en grupo de dos):

- Un estudiante introduce su calificación, si es menor que 11 tiene que imprimir "Desaprobaste".

Pseudocódigo:	Diagrama de flujo:	Programa en Turbo Pascal
		Resultado

- Un hombre introduce su edad, si es mayor que 18 tiene que imprimir "eres mayor de edad"

Pseudocódigo:	Diagrama de flujo:	Programa en Turbo Pascal:
		Resultado

- Un estudiante introduce su edad, si es menor que 18 tiene que imprimir “eres menor de edad”.

Pseudocódigo:	Diagrama de flujo:	Programa en Turbo Pascal:
		Resultado

ESTRUCTURA DE CONTROL if

La estructura de control de selección principal, es una sentencia if. Los formatos principales que tiene son:

```
if (expresión)Thensentencia;
```

```
if (condición) Then sentencia
else
sentencia;
```

Se utiliza la sentencia if dentro de un programa, se evalúa la expresión entre paréntesis que viene a continuación de if. Si la expresión o condición es verdadera, se ejecuta la acción o sentencia, si es falsa la ejecución del programa continua con la siguiente sentencia programada o finaliza el programa.

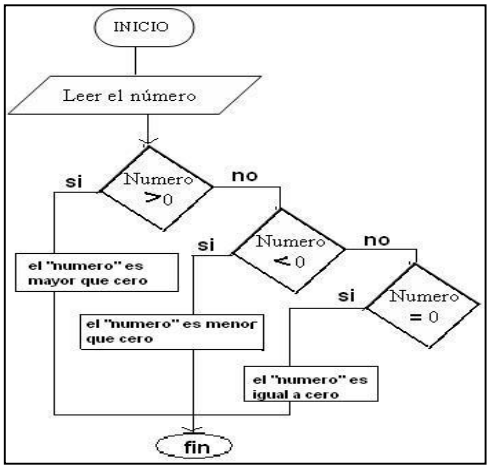
Ejemplo:

Deseamos comparar un número negativo o positivo, que imprima si es mayor, menor o igual a cero

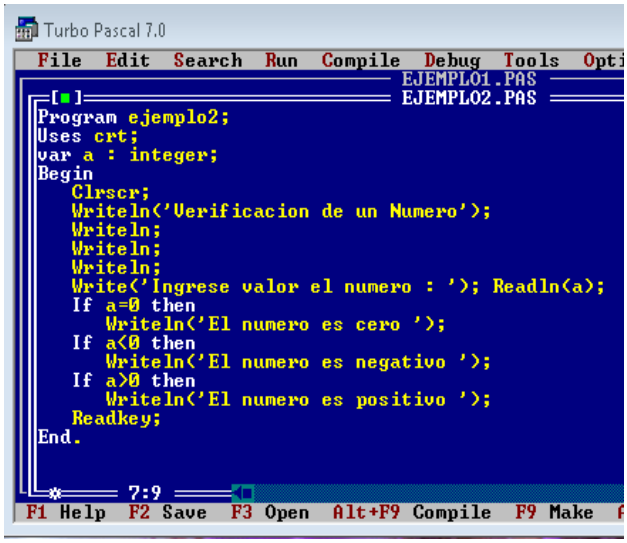
1) Pseudocódigo:

```
ifnumero es mayor a cero
printf es mayor que cero
ifnumero es menor que cero
printf es menor que cero
if es igual a cero
printf es igual a cero
```

2)Diagrama de flujo:



3) Digitamos siguiendo la estructura del lenguaje C:



4) Corremos el programa con CTRL + F5y si hay algún error nos dirá y no mostrara ningún resultado.

Resultados:

Introduzca un número positivo o negativo: 6
6.000 es mayor que cero
Introduzca un número positivo o negativo: -7
-7.000 es mayor que cero
Introduzca un número positivo o negativo: 0
0.000 es igual que cero

Síntesis

El procesamiento de información transforma datos primarios o información de base en información final con los requisitos que debe de tener, basados en un esquema simple como un proceso productivo donde el ingreso de materia prima equivaldría al Input, luego la transformación de esta seria el procesamiento de datos y por último el producto terminado es el Output, es decir se transforman los datos mediante el procesamiento automático apoyados por el computador y como resultado la información.

El conjunto de instrucciones que indican los pasos a realizar, en orden, para resolver un sistema específico o clase de problemas, se denomina algoritmo.

El algoritmo es un conjunto de pasos que resuelven un problema en forma manual o por métodos mecanizados.

Para crear o desarrollar un software es necesario sistematizar mediante una metodología denominada Ciclo de desarrollo del Software el cual en forma gradual permite la implementación del Software.

Apreciacion crítica y sugerencia

A lo que se quiere llegar con la monografía de ELEMENTOS A LA PROGRAMACION es que los alumnos y estudiantes de todo nivel por lo menos tengan una noción o una base antes de empezar a trabajar con diferente tipo de lenguajes de programación.

Muy aparte de esto lo que quizás nos manifiesta es la gran importancia que tiene este trabajo ya que da el camino para que el alumno sea el propio creador de un software sea Educativo o no.

Así pues lo que se debe tener en cuenta son los conceptos proporcionados en este trabajo.

Como vemos que comenzamos con una breve introducción de algunos conceptos claves para entender la programación, así pues pasamos a uno de los más básicos los Fundamentos de Programación con lo que después ya el alumno tenga la necesidad de buscar herramientas que faciliten el uso pero siempre teniendo en cuenta las fases del diseño de Programación tomando muy en cuenta la metodología de PROGRAMACION, PARA FINALMENTE CREAR NUESTRA APLICACIÓN DE CODIGO.

Ahora bien la aplicación didáctica debe considerarse que el alumno llegue a captar al máximo y que del mismo tratar de investigar y ver cuanto el sabe de este tema y como ha creado su propio significado personal.

Referencias

Libros:

Cairo **O., METODOLOGIA DE LA PROGRAMACION** 2da edición **Editorial:** ***Alfaomega 2013***

Cibertec **LENGUAJE DE PROGRAMACION I Editorial:** ***Universidad P. de Ciencias Aplicadas S.A.C 2008***

Joyanes L., ***ALGORITMIA:*** 3era Edición. **Editorial**: ***Concepción Fernández Madrid Mc Graw Hill 2013***

Izquierdo ***L., INTRODUCCION A LA PROGRAMACION ORIENTADA A OBJETOS,*** **Editorial:** ***Concepción Fernández Madrid Mc Graw Hill***

Santo M., - Patiño I., ***FUNDAMENTOS DE PROGRAMACION E*****ditorial:** ***San Marcos 2016***

Monografías:

Casariego More, Edwin **INTRODUCCION A LA PROGRAMACION 2010**

Tapara Condo, Percy **INTRODUCCION A LA PROGRAMACION 2010**

Páginas Web

- www.programacion.com/.../introduccion_a_la_programacion_205
- http://www.lenguajes-de-programacion.com/herramientas-de-programacion.shtml
- www.algoritmia.net/articles.php?id=55
- www.programacionfacil.com