UNIVERSIDAD CONTINENTAL

MODALIDAD VIRTUAL

PROGRAMACIÓN I

Datos de catalogación bibliográfica

Carol R. Rojas Moreno
PROGRAMACIÓN I

Huancayo: Fondo Editorial de la Universidad Continental, 2015.

ISBN:
Materia: PROGRAMACIÓN I

Formato 21x29,7 cm. Páginas: 166

PROGRAMACIÓN I / AULA VIRTUAL

Primera edición: febrero 2016

Director: Emma Barrios Ipenza
Editor: Eliana E. Gallardo Echenique
Diseñador didáctico:
Diseño gráfico: Francisco Rosales Guerra

PROGRAMACIÓN I
Autor: Carol R. Rojas Moreno
Jefatura de Virtualización de Contenidos

ÍNDICE

INTRODUCCIÓN

La programación en computadores es desarrollada en diferentes lenguajes de programación, pero manteniendo los principios básicos para su realización. Aplicar los conceptos fundamentales de programación en computadores ha permitido la elabo-ración de aplicaciones de software para las organizaciones, otorgando soluciones informáticas para sus procesos.

Los procesos de una organización, están siendo automatizadas, desde pequeñas a grandes empresas, por lo que dar soluciones a procesos relativamente sencillos, pero claves a través de la programación en computadoras, constituirá la meta de aprendizaje de esta asignatura.

Programación I es una asignatura que proporcionará al estudiante los conocimientos necesarios en la elaboración de programas basados en el enfoque estructurado, requeridos en su formación básica para desarrollar programas en futuros niveles más avanzados.

Los contenidos propuestos en este material de estudio están divididos en cuatro unidades, que son: Unidad I "Conceptos básicos de programación"; Unidad II "Estructuras de control para la programación: Selectivas"; Unidad III "Estructuras de control para la programación: Repetitivas"; y Unidad IV "Módulos para la programación: funciones y procedimientos".

El Manual autoformativo presenta diversos temas que buscan que el estudiante adquiera los conocimientos necesarios para elaborar programas en computadoras, considerando la definición y representación de las Estructuras de control para la programación, además de segmentar un programa complejo, en programas más simples y ser reutilizados en otros programas, para manipular y operar los datos, conceptos que serán usados en las diferentes áreas a lo largo de su formación profesional.

DIAGRAMA DE PRESENTACIÓN DE LA ASIGNATURA

RESULTADO DE APRENDIZAJE:

Al finalizar la asignatura, el estudiante será capaz de elaborar programas computacionales, en un lenguaje de programación, aplicando las estructuras de control y módulos de programa como propuesta de solución a un problema del entorno o de la realidad.

UNIDADES DIDACTICAS

UNIDAD I	UNIDAD II	UNIDAD III	UNIDAD IV
CONCEPTOS BÁSICOS DE PROGRAMACIÓN	ESTRUCTURAS DE CONTROL PARA LA PROGRAMACIÓN: SELECTIVAS	ESTRUCTURAS DE CONTROL PARA LA PROGRAMACIÓN: REPETITIVAS	MÓDULOS PARA LA PROGRAMACIÓN: FUNCIONES Y PROCEDIMIENTOS

TIEMPO MÍNIMO DE ESTUDIO

UNIDAD I	UNIDAD II	UNIDAD III	UNIDAD IV
1ra. Semana y 2da. Semana 16 Horas	3ra. Semana y 4ta. Semana 16 Horas	5ta. Semana y 6ta. Semana 16 Horas	7ma. Semana y 8va. Semana 16 Horas

UNIDAD I

CONCEPTOS BÁSICOS DE PROGRAMACIÓN.

DIAGRAMA DE PRESENTACIÓN DE LA UNIDAD I

Al finalizar la unidad, el estudiante será capaz de diseñar algoritmos, considerando las características de: lógico, ordenado y finito, usando los elementos del Diagrama de flujo.

CONTENIDOS	ACTIVIDADES FORMATIVAS (HABILIDADES Y ACTITUDES)	SISTEMA DE EVALUACIÓN (TÉCNICAS Y CRITERIOS)
Tema Nº 1 : Algoritmo 1 Definición 2 Características 3 Instrucciones algorítmicas básicas. **Tema Nº 2 : Representación del algoritmo:** 1 Pseudocódigo 2 Diagrama de flujo **Tema Nº 3 : Programación estructurada** 1 Programa 2 Lenguaje de programación 3 Programas traductores 4 Definición de Programación estructurada.	• Lee y analiza el tema Algoritmo. Observa el vídeo Tutorial de C++ en Español # 4 - Tipos de datos y elabora un cuadro de información con los tipos de datos, según las instrucciones algorítmicas básicas. • Lee el tema de Representación de algoritmo. Observa el vídeo Pseudocódigo y Diagrama de Flujo | Iniciándose en la programación - 03 - Tutoriales y más, y los compara elaborando una sencilla representación del desarrollo de programas según la lectura N° 1. • Identifica los elementos de representación de los diagramas de flujo y sus características y elabora un diagrama para los casos propuestos.	**Procedimientos e indicadores de evaluación permanente:** • Entrega puntual de trabajos realizados. • Calidad, coherencia y pertinencia de contenidos desarrollados. • Prueba teórico-práctica, individual. • Actividades desarrolladas en sesiones tutorizadas. **Criterios de evaluación para Diagrama de representación:** • Cuadro de identificación de partes del algoritmo: entrada, proceso, salida. • Diagrama de flujo con el uso de las instrucciones algorítmicas básicas.

RECURSOS:

VIDEOS:

Tema Nº 1: Tutorial de C++ en Español # 4 - Tipos de datos

https://www.youtube.com/watch?v=vqLFK_CYDb4

Tema Nº 2: Pseudocódigo y diagrama de flujo | Iniciándose en la programación - 03 - Tutoriales y más

https://www.youtube.com/watch?v=CfaKRlOvfgA

Tema Nº 3: Diagrama de flujo con PSeint

https://www.youtube.com/watch?v=Urf3s3Wz5Eo

DIAPOSITIVAS ELABORADAS POR EL DOCENTE:

Lectura complementaria:

Lectura Seleccionada Nº 1

Gallardo, J. y García, C. Diseño de algoritmos y programas.Dpto. Lenguajes y Ciencias de la Computación – UMA.

INSTRUMENTO DE EVALUACIÓN	Lista de cotejo N° 1
BIBLIOGRAFÍA (BÁSICA Y COMPLEMENTARIA)	**BASICA** Joyanes, L. (2008) *Fundamentos de programación*. Cuarta Edición. España : McGRAW-HILL. **COMPLEMENTARIA** Marcelo, R. (2014). *Fundamentos de programación C++*. Lima: Editorial Macro. Suarez, R. G. (2003). *Algoritmos y diagramas de flujo*. Lima: Libros y Publicaciones.
RECURSOS EDUCATIVOS DIGITALES	Gallardo, J. (1999). Dpto. de Lenguajes y Ciencias de la Computación UMA. Recuperado en iciembre de 2015, de: http://www.lcc.uma.es/~pepeg/modula/temas/tema2.pdf Tecnología, A. (20 de Junio de 2015). Area Tecnología. Obtenido de: http://www.areatecnologia.com/informatica/ejemplos-de-diagramas-de-flujo.html

TEMA N° 1: ALGORITMO

Estimado estudiante, para esta primera unidad se presenta los conceptos básicos de programación, como la definición y las características de un algoritmo, considerando las instrucciones algorítmicas y los elementos de representación, para que pueda iniciar en la elaboración de programas.

Por lo que, estos conceptos serán ejemplificados en situaciones de la realidad que usted conoce, para que pueda asociar los conceptos de programación con sus experiencias del entorno.

1. DEFINICIÓN

(Luis Joyanes Aguilar, 2008), La palabra algoritmo se dio en honor del matemático persa del siglo IX, Khowârizmî.

Una definición conocida de algoritmo es la de ser un conjunto ordenado y finito de tareas (instrucciones si se ejecutan en un computador, algoritmo computacional) que conducen a la solución de un problema.

Por ejemplo:

- Las actividades que una persona realiza desde que sale de su casa para llegar a su centro de labores.
- Los pasos que realiza para encender su equipo de cómputo.
- Las actividades que realiza para realizar la compra de un producto por internet.

Un algoritmo computacional se expresa con código o instrucciones en un determinado lenguaje de programación, teniendo como resultado final, un programa.

Se identifican las partes de un algoritmo como: Entrada, Proceso y Salida.

Figura N° 1: Partes de un algoritmo
Fuente: Carol Rojas M.

A continuación, le mostramos dos ejemplos de algoritmo y recurrimos a la identificación de sus partes para su solución:

Ejemplo 1:

Se requiere realizar los pasos para sumar dos números naturales, teniendo como salida el valor del cálculo:

ENTRADA DATOS A USAR	PROCESO (PROCEDIMIENTO DE CÁLCULO)	SALIDA DATOS PARA REPORTAR
Numero1 Numero2	Suma = Número1 + Número2	Suma

Cuadro N° 1: Partes de un algoritmo: Ejemplo Suma
Fuente: Carol Rojas M.

Ejemplo 2:

Se requiere realizar los pasos para ingresar un producto para la venta, y verificar si existe, por lo que requerimos como salida un mensaje de disponibilidad y si desea realizar otra acción:

ENTRADA DATOS A USAR	PROCESO (PROCEDIMIENTO DE CÁLCULO)	SALIDA DATOS PARA REPORTAR
Código de producto	Si (Código Producto es igual a CodProductoBD), Entonces, Mostrar "Producto Disponible" Mostrar "¿Desea Comprar?" Sino, "No existe Producto"	"Producto Disponible", "¿Desea comprar?"

Cuadro N° 2: Partes de un algoritmo: Ejemplo Buscar
Fuente: Carol Rojas M.

2. CARACTERÍSTICAS

Luis Joyanes Aguilar presenta las características principales de un algoritmo: [1]

- "Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar bien definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; es decir, debe tener un número finito de pasos."

En Fundamentos de Programación, Ricardo Marcelo Villalobos nos ofrece las siguientes características: [2]

Características que deben cumplir los algoritmos obligatoriamente:

- Un algoritmo debe resolver el problema para el que fue formulado. Lógicamente, no sirve un algoritmo que

[1] Joyanes, L. (2008). Fundamentos de programación. España: McGRAW-HILL.

[2] Marcelo, R. (2014). Fundamentos de programación C++. Lima: Editorial Macro.

no resuelve ese problema. En el caso de los programadores, a veces crean algoritmos que resuelven problemas diferentes a los planteados.

- Los algoritmos son independientes al lenguaje de programación. Los algoritmos se escriben para ser usados en cualquier lenguaje de programación.
- Los algoritmos deben ser precisos. Los resultados de los cálculos deben ser exactos, de manera rigurosa. No es válido un algoritmo que solo aproxime la solución.
- Los algoritmos deben ser finitos, deben finalizar en algún momento. No es un algoritmo válido aquel que produce situaciones en que el algoritmo no termina.
- Los algoritmos deben poder repetirse, deben permitir su ejecución las veces que haga falta. No son válidos lo que tras ejecutarse una vez ya no pueden volver a hacerlo por la razón que sea.

Características aconsejables de los algoritmos:

- **Validez:** Un algoritmos es válido si carece de errores. Un algoritmo puede resolver el problema para el que se planteó y, sin embargo, no ser válido por que posee errores.
- **Eficiencia:** Un algoritmo es eficiente si obtiene la solución al problema en poco tiempo. No lo es si tarda en obtener el resultado, es decir, se debe usar los recursos requeridos, ni mas ni menos.
- **Óptimo:** Un algoritmo es óptimo si es el más eficiente posible y no contiene errores. La búsqueda de este algoritmo es el objetivo prioritario del programador. No siempre podemos garantizar que el algoritmo hallado sea el óptimo, a veces sí.

Pseudocódigo:

```
INICIO
   Levante la bocina
   Espere tono
   Marque el número
   Espere que contesten
   Hable con la otra persona
   Cuelgue la bocina
FIN
```

Diagrama de flujos:

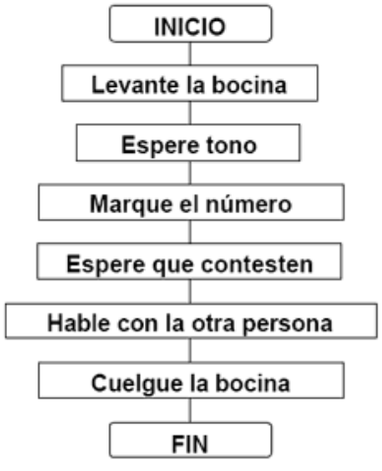


Figura N° 2 Ejemplo de un algoritmo
Fuente: http://www.desarrolloweb.com/articulos/2199.php

3. INSTRUCCIONES ALGORÍTMICAS BÁSICAS

Para elaborar un algoritmo y, sobre todo, si se va a escribir en un lenguaje de programación, debemos considerar el uso de variables en cada una de las partes del algoritmo.

VARIABLE

Con seguridad, usted utiliza esta palabra en su día a día como incógnita, un dato que varía, o la representación de un valor, y aquí en la programación no es diferente.

Lo que se debe considerar en la programación es que una variable corresponde a un espacio de memoria del computador y, por lo tanto, referenciado por una dirección de memoria (base hexadecimal).

Este espacio de memoria almacena lo valores que ingresemos o que se procesan y que se reportan.

Estas variables, para que puedan ser usadas en un lenguaje de programación, deben ser definidas con un tipo de dato y luego un nombre, recomendándose que este nombre sea un identificador o de nomenclatura que represente el significado propio de la variable.

Gráficamente, una variable es representada como una casilla con datos y dirección de memoria.

Figura N° 3: Representación de una variable
Fuente: Carol Rojas M.

Ejemplos:

- Entero CantidadProducto
- Cadena NombreCliente
- Real peso, talla

Tipos de datos básicos (simples predefinidos) ***en C:***

Numéricos:
Entero (`int`)
Real (`float` y `double`)
Carácter (`char`)
Sin valor (`void`)

© carlospes.com

Figura N° 4: Ejemplo de tipos de datos en C++
Fuente: http://manualborland30.blogspot.pe/p/tipos-de-datos.html

Según el lenguaje de programación que se esté utilizando, los tipos de datos son similares, variando la sintaxis o escritura del tipo de dato y el tamaño que ocupa en el espacio de memoria.

.NET Framework	VB 2010	VB6
System.Boolean	Boolean	Boolean *
System.Byte	Byte	Byte
System.Int16	Short	Integer
System.Int32	Integer	Long
System.Int64	Long	N.A.
System.Single	Single	Single
System.Double	Double	Double
System.Decimal	Decimal	Currency *
System.Char	Char	N.A. (ChrW)
System.String	String	String *
System.Object	Object	Variant / Object *
System.DateTime	Date	Date *
System.SByte	SByte	N.A.
System.UInt16	UShort	N.A.
System.UInt32	UInteger	N.A.
System.UInt64	ULong	N.A.

Figura N° 5: Tipos de datos en Visual Basic
Fuente: https://www.programarya.com/Cursos/Visual-Basic/Sistema-de-Tipos/Tipos-Primitivos

Recuerde que para la presente asignatura se usará un lenguaje de programación bajo el enfoque estructurado denominado Lenguaje C y que permitirá enfocar su aprendizaje solamente en la codificación del algoritmo.

Tipo	Ancho en Bits	Rangos mínimos en 16 bits
char	8	[-128,127]
unsigned char	8	[0,255]
int	16	[-32768,32767]
unsigned int	16	[0,65.535]
short int	8	[-128,127]
unsigned short int	8	[0,255]
long int	32	[-2147483.648,2147483647]
unsigned long int	32	[0,4294967295]
long long int	64	[−9223372036854775808, 9223372036854775807]
unsigned long long int	65	[0, 18446744073709551615]
float	32	[-3.4E38,-1.2E-38],0, [1.2E-38,3.4E38]
double	64	[-1.7E308,-2.3E-308],0,[2.3E-308,1.7E308]
long double	128	[-1.1E4932,-3.4E-4932],0,[3.4E-4932,1.1E4932]

Figura N° 6: Tipos de datos en C++
Fuente: http://www.vivedev.es/2015/02/manipulacion-de-datos-en-c-c-parte-2.html

A continuación, se presenta un ejemplo para sumar dos números, considerando los tipos de datos de las variables a usar.

VARIABLES A USAR	ENTRADA DATOS A USAR	PROCESO (PROCEDIMIENTO DE CÁLCULO)	SALIDA DATOS PARA REPORTAR
Entero numero1 Entero numero2 Entero Suma	numero1 numero2	Suma = numero1 + numero2	Suma

Cuadro N° 3: Variables y tipos de datos: Ejemplo Suma
Fuente: Carol Rojas M.

Las instrucciones algorítmicas básicas que usan variables son:

a. Entrada: acción de ingresar un dato (valor) a una **variable,** se puede expresar con el pseudocódigo **LEER**.

Ejemplo:

LEER edad

b. Salida: acción de mostrar un dato (valor) de una **variable,** se expresa en el pseudocódigo **MOSTRAR**.

Ejemplo:

MOSTRAR TotalCompra

c. Asignación: acción de dar a una **variable** el valor de una **expresión**. Donde **variable** y el valor de **expresión** deben tener el mismo tipo de dato.

Ejemplo:

resultado numero1 + numero 2

```
// Ingresar Valores
    cin>>Variable1;
    cin>>Variable2;

// Procesar
    Total = Variable 1 + Variable2;

// Mostrar Mensajes y Valores
    cout<<"El resultado es: ";
    cout<<Total;
```

Figura N° 7: Instrucciones algorítmicas en C++
Fuente: Carol Rojas M.

Además, existen operadores que ayudan a las variables a ingresar, calcular o mostrar su información.

Estos operadores se pueden clasificar en operadores de asignación, lógicos, aritméticos, relacionales y se usan su jerarquía en el programa.

Operadores Aritméticos			
Operador	**Descripción**	**Ejemplo a=3, b=2**	**Resultado**
+	suma	a+b	5
-	resta y valor negativo (unario)	a-b -b	1 -2
*	multiplicación	a*b	6
/	división	a/b	1
%	resto	a%b	1

Operadores Relacionales			
Operador	**Descripción**	**Ejemplo**	**Resultado**
>	mayor que	2 > 3	0
<	menor que	2 < 3	1
>=	mayor o igual que	2 >= 3	0
<=	menor o igual que	2 <= 3	1
==	igual	2 == 3	0
!=	distinto	2 != 3	1

Operadores Lógicos			
Operador	**Descripción**	**Ejemplo a=1, b=2, c=3**	**Resultado**
&&	y (and)	(a > b) && c	0
||	o (or)	(c > a) || c	1
!	no (not)	!(a >c)	1

Operadores de Asignación		
Operador	**Ejemplo**	**Equivalencia**
=	a=b=c	a=c; b=c;
+=	a+=3	a=a+3
-=	a-=3*b	a=a-(3*b)
*=	a*=2	a=a*2
/=	a/=35+b	a=a/(35+b)
%=	a%=8	a=a%8
>>=	a>>=1	a=a>>1
<<=	a<<=b	a=a<<b
&=	a &=(c+=3)	c=c+3; a=a&c
^=	a ^= 2	a = a^2
|=	a |= c	a = a | c

Figura N° 8 Tipos de operadores en C++
Fuente: http://serdis.dis.ulpgc.es/~itie-fi/Programacion/C/apuntesc/Operadores.html

También se presenta un ejemplo para calcular el monto total de pago, según la cantidad de productos a vender y la asignación de descuentos, considerando los tipos de datos de las variables a usar.

VARIABLES A USAR	ENTRADA DATOS A USAR	PROCESO (PROCEDIMIENTO DE CÁLCULO)	SALIDA DATOS PARA REPORTAR
Cadena NombProd Entero CantProd Real PrecProd Real MontoBruto Real PorcDscto Real MontoDscto Real MontoTotalPago	NombProd CantProd PrecProd	MontoBruto = CantProd * PrecProd Si (CantProd <= 10) PorcDscto = 0.2 Sino Si (CantProd > 10) PorcDscto = 0.4 MontoDscto = MontoBruto * PorcDscto MontoTotalPago = MontoBruto – MontoDscto	MontoBruto MontoDescuento MontoTotalPago

Cuadro N° 4: Variables y tipos de datos: Ejemplo Calcular monto de pago
Fuente: Carol Rojas M.

SÍNTESIS DEL TEMA I

ALGORITMO

- Conjunto de instrucciones en orden lógico y finito.
- Tiene tres partes: ENTRADA, PROCESO, SALIDA.
- Usa instrucciones algorítmicas y variables.

ACTIVIDAD FORMATIVA Nº 1

Lee y analiza el tema Algoritmo. Complemente la información observando el vídeo "Tutorial de C++ en español # 4 - Tipos de datos" y elabora un cuadro de información con los tipos de datos, según las instrucciones algorítmicas básicas.

INSTRUCCIONES:

1. Lee y analiza el tema N° 1 y extrae las ideas fundamentales del algoritmo en sus características e instrucciones algorítmicas básicas.
2. Observe el vídeo **"Tutorial de C++ en español # 4 - Tipos de datos"** y complemente la información obtenida en el tema N°1.
3. Elabore un cuadro de información con los tipos de datos sobre los Tipos de datos, variables, constantes en C/C++.

TIPO DE DATO	SINTAXIS (ESCRITURA EN LENGUAJE C)	DESCRIBA UN EJEMPLO EN EL CUAL USTED RECOMENDARÍA SU USO. (DIFERENCIÁNDOLO DE LOS OTROS TIPOS DE DATOS)
Entero		
Float		
Double		
Char		
String		
Boolean		

VARIABLE ¿QUÉ ES?	VARIABLE SINTAXIS EN EL C++:	DESCRIBA UN EJEMPLO EN EL CUAL USTED RECOMENDARÍA SU USO
CONSTANTE ¿QUÉ ES?	**CONSTANTE SINTAXIS EN EL C++:**	**DESCRIBA UN EJEMPLO EN EL CUAL USTED RECOMENDARÍA SU USO**

TEMA N° 2: REPRESENTACIÓN DEL ALGORITMO

Estimado estudiante, dado el título de este tema, seguro que estará imaginando una representación tediosa para su algoritmo.

Recuerde que el objetivo de una modelo de representación es ayudar a comprender el proceso, y no complicarlo.

Por lo que, a continuación, y basados en el libro de Luis Joyanes Aguilar (2008), se muestran algunos modelos para representar el algoritmo y que serán usados en la siguiente unidad.

Observará que cada una de las representaciones, mantiene las tres partes del algoritmo, y la característica de finito, representado por los símbolos de inicio y fin en cada ejemplo.

1. PSEUDOCÓDIGO

Es la representación del algoritmo aproximándose al lenguaje natural de quien elabora el programa.

Es de fácil comprensión y modificación, ya que no requiere de otros símbolos para representar cada uno de los pasos de solución a un problema.

Recuerde considerar las partes del algoritmo para que le ayude a elaborar dicho algoritmo con sus respectivas características.

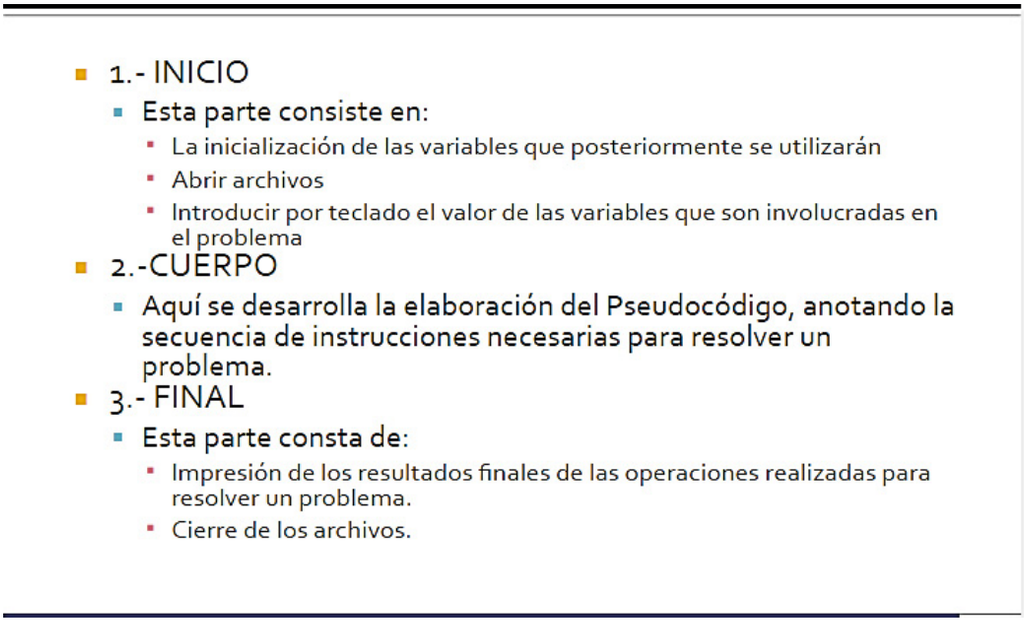


Figura N° 9: Estructura de pseudocódigo
Fuente: http://slideplayer.es/slide/2315600/

Ejemplo: sumar dos números enteros.

```
INICIO
entero          numero1, numero2, resultado
Leer  numero1
Leer  numero2
resultado           numero1 + numero 2
Escribir  "El resultado de la suma es: "
Escribir  resultado
FIN
```

A continuación, se muestra un ejemplo de ingresar el resultado de la calificación de un estudiante, y el algoritmo cuenta la cantidad de aprobados y la cantidad de desaprobados.

```
Inicio
     aprobados ← 0
     reprobados ← 0
     resultado ← 0
     estudiantes ← 1
     Mientras estudiantes <= 10
          Leer resultado
          Si resultado == 1 entonces
               aprobados ← aprobados + 1
          SiNo
               reprobados ← reprobados + 1
          FinSi
          estudiantes ← estudiantes + 1
     FinMientras
     Mostrar aprobados
     Mostrar reprobados
     Si aprobados > 8 entonces
          Mostrar "Aumentar la colegiatura"
     FinSi
Fin
```

Figura N° 10: Ejemplo de algoritmo usando contador
Fuente: http://www.casdreams.com/auladeinformatica/cet/algoritmos.htm

Otro ejemplo de algoritmo, para evaluar si un número natural es primo.

```
entero num, x, resto

inicio
  leer (num)
  x ← 2
  resto ← num mod x
  mientras ( (resto < > 0) and (x < num) )
    hacer
      x ← x + 1
      resto ← num mod x
    fin hacer
  si (x = num)
    entonces escribir ("El número es primo")
    si no     escribir ("El número no es primo")
  fin si
final
```

Figura N° 11: Ejemplo de algoritmo para determinar número primo
Fuente: http://henostrozagutierrezomarchristian.blogspot.pe/2015_10_01_archive.html

2. DIAGRAMA DE FLUJO

Representación con símbolos de entrada y procesos, usando flujos entre los procesos a realizar. Los principales símbolos de representación también consideran la entrada, salida y proceso del algoritmo: [3]

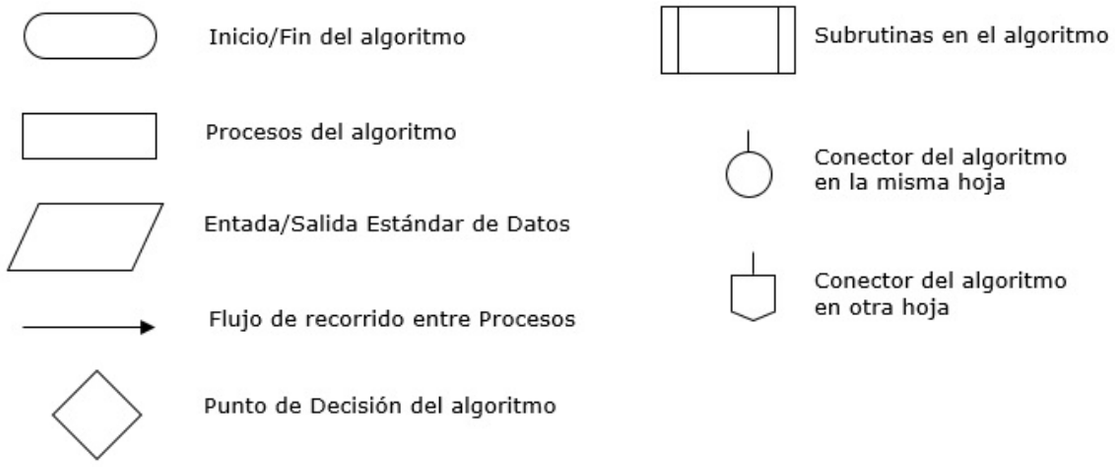


Figura N° 12: Símbolos más usados del Diagrama de flujo
Fuente: Carol Rojas M.

Existen otros símbolos que no son usados en la actualidad debido a los cambios en los lenguajes de programación y mecanismos de entradas y salidas de datos.

SÍMBOLO	REPRESENTA	SÍMBOLO	REPRESENTA
*	Operación con teclado. Representa una operación en que se utiliza una perforadora o verificadora de tarjeta.		Dirección de flujo o línea de unión. Conecta los símbolos señalando el orden en que se deben realizar las distintas operaciones.
*	Tarjeta perforadora. Representa cualquier tipo de tarjeta perforada que se utilice en el procedimiento.	*	Cinta magnética. Representa cualquier tipo de cinta magnética que se utilice en el procedimiento.
*	Cinta perforada. Representa cualquier tipo de cinta perforada que se utilice en el procedimiento.	*	Teclado en línea. Representa el uso de un dispositivo en línea para promocionar información a una computadora electrónica u obtenerla de ello.

Figura N° 13: Otros símbolos del Diagrama de flujo
Fuente: http://es.slideshare.net/AliniuZizRguezT/simbolos-diagrama-de-flujo

[3] Suarez. G. (2003). Algoritmos y diagramas de flujo. Lima: Editorial Libros y Publicaciones.

Ejemplo de diagrama de flujo para una estructura de control selectiva múltiple:

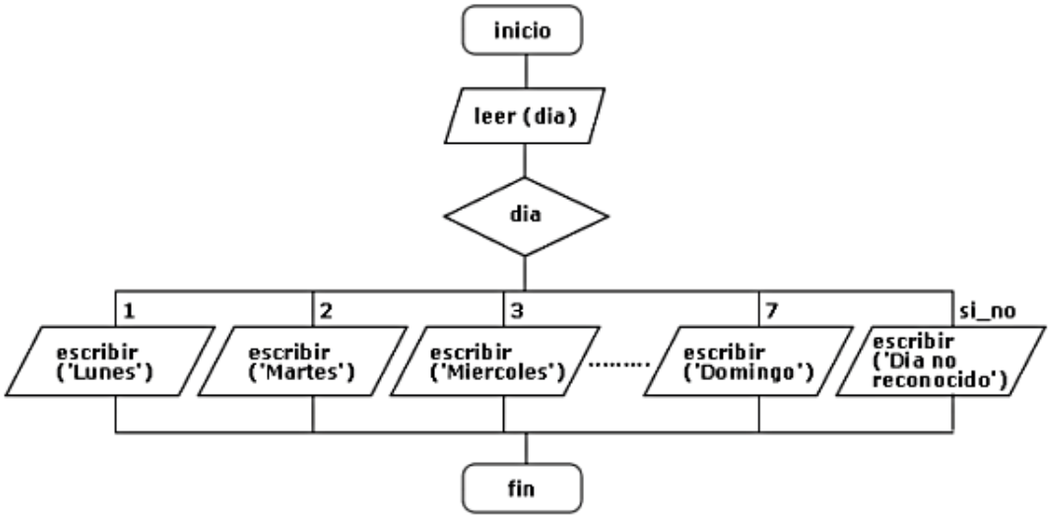


Figura N° 14: Ejemplo de Diagrama de flujo selectivo
Fuente: https://elianahernandez.files.wordpress.com/2013/03/estructura-selectiva-case-teoria.pdf

Ejemplo de diagrama de flujo para una estructura de control repetitiva, que se revisará en las próximas unidades:

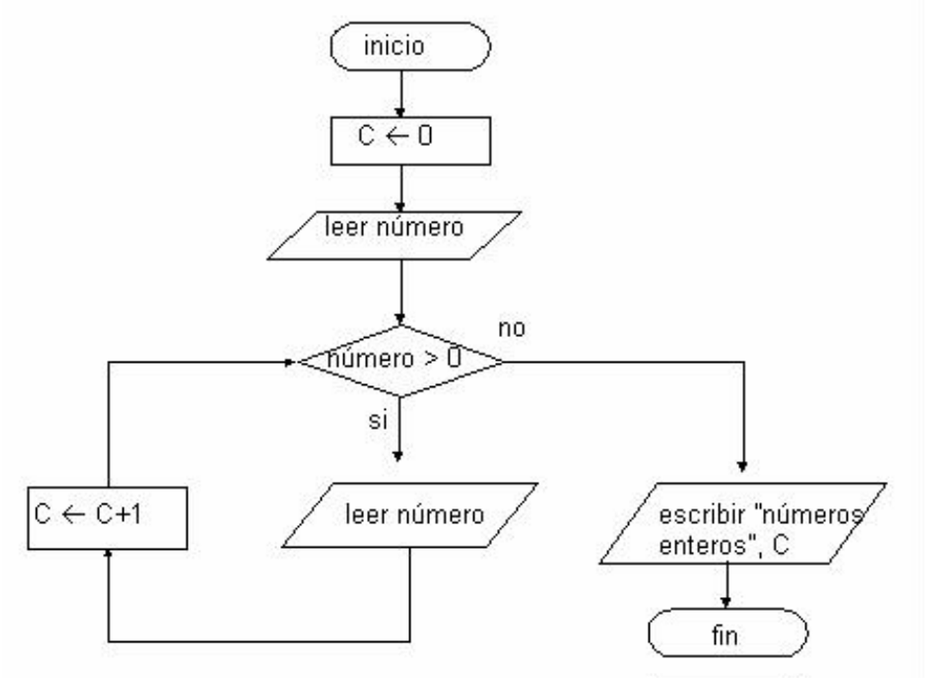


Figura N° 15: Ejemplo de Diagrama de flujo repetitivo
Fuente: http://daamideas.blogspot.pe/2015/05/documentos-comerciales-lo-s-documentos.html

Existen herramientas que permiten elaborar un diagrama de flujo, además de ofrecer instrucciones algorítmicas, tales como:

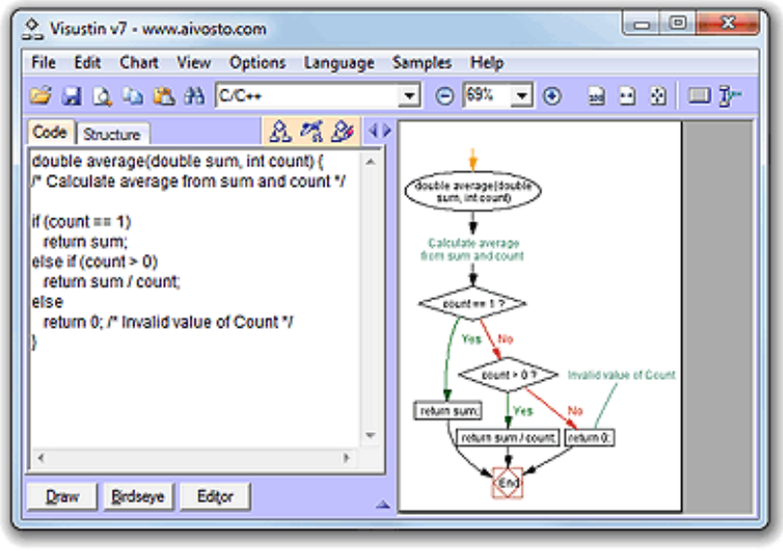


Figura N° 16: Visustin – Generador de diagramas de flujo
Fuente: http://www.aivosto.com/visustin-es.html

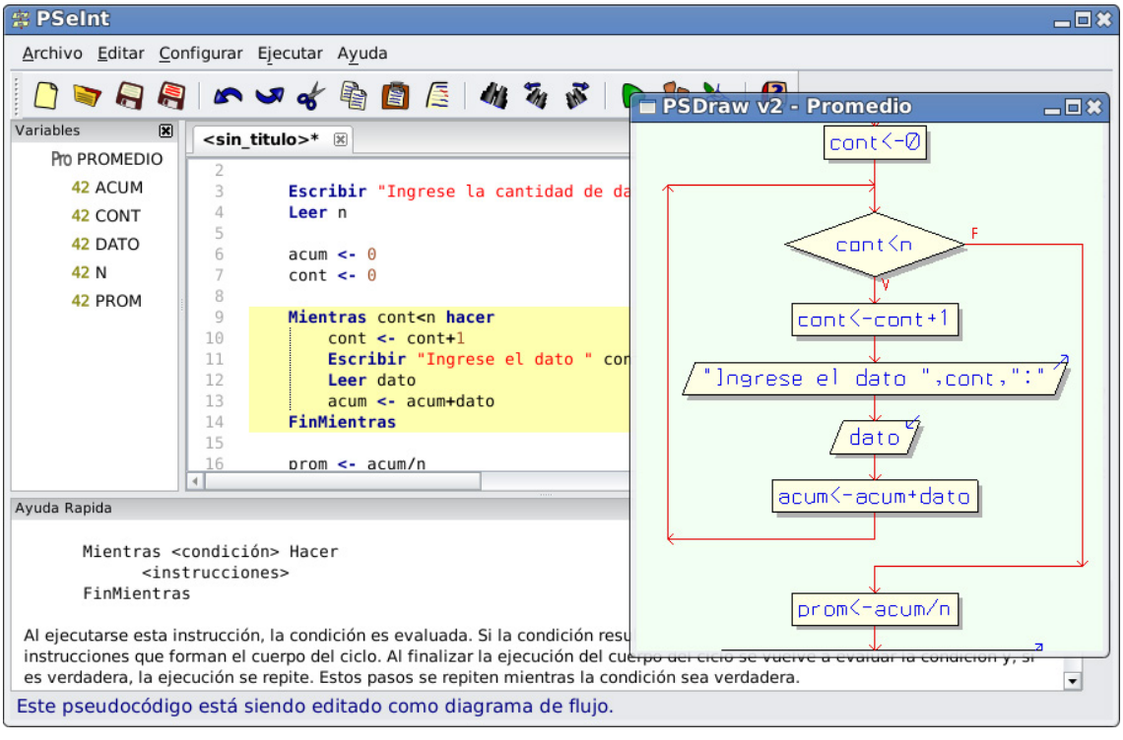


Figura N° 17: PSeInt – Generador de diagramas de flujo
Fuente: http://ivonmodalidadsistemas10-03jm.blogspot.pe/2015/06/pseint-definicion-pseint-es-una.html

Estimado estudiante, adicionalmente puede encontrar información de otra forma de representación que simplifica el uso de símbolos, pero no permite evaluar con claridad el flujo o secuencia de actividades a realizar, sobre todo si estamos en un nivel inicial en la programación, esta representación se detalla a continuación:

Diagrama Nassi/Schneiderman (N-S)

Representación en bloques, es decir, cada uno de los procesos como ingreso/salida de datos, decisiones, acciones, repeticiones.

Es decir, el resultado de la ejecución de un bloque, representa la entrada para el siguiente bloque.

Mostramos un ejemplo simple de un Diagrama N-S, en el cual se calcula con los daros previamente inversados.

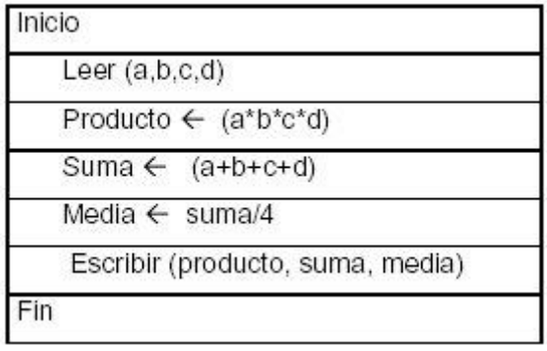


Figura N° 18: Ejemplo Simple de Diagrama de N-S
Fuente: http://www.mailxmail.com/curso-aprende-programar/representacion-grafica-algoritmos

Ejemplo de Diagrama de bloques para determinar el mayor y menor de dos números:

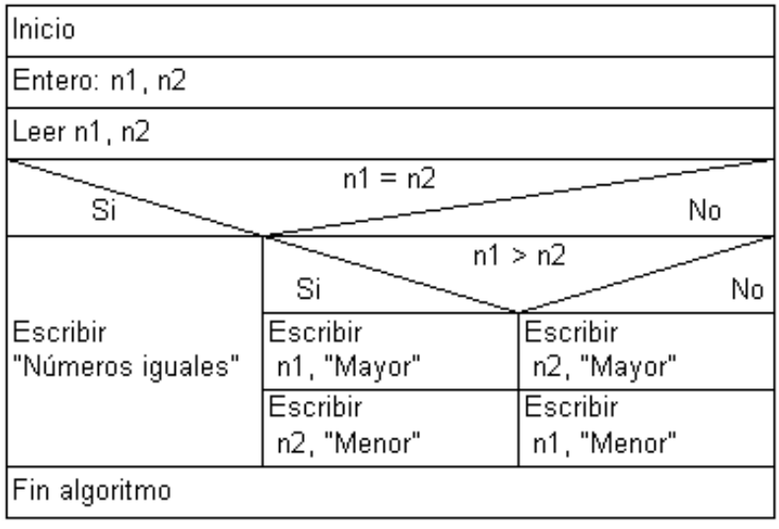


Figura N° 19: Ejemplo condicional de Diagrama de N-S
Fuente: http://www.monografias.com/trabajos19/algoritmos/algoritmos.shtml

Para el desarrollo de software, se hace uso de representación, pero con notaciones más avanzadas como es UML (Unified Modeling Language) o BPMN (Business Process Model Notation), que permiten diseñar el algoritmo de solución que proporcionará el software.

Observe la similitud de los diagramas mostrados, con el diagrama de flujo.

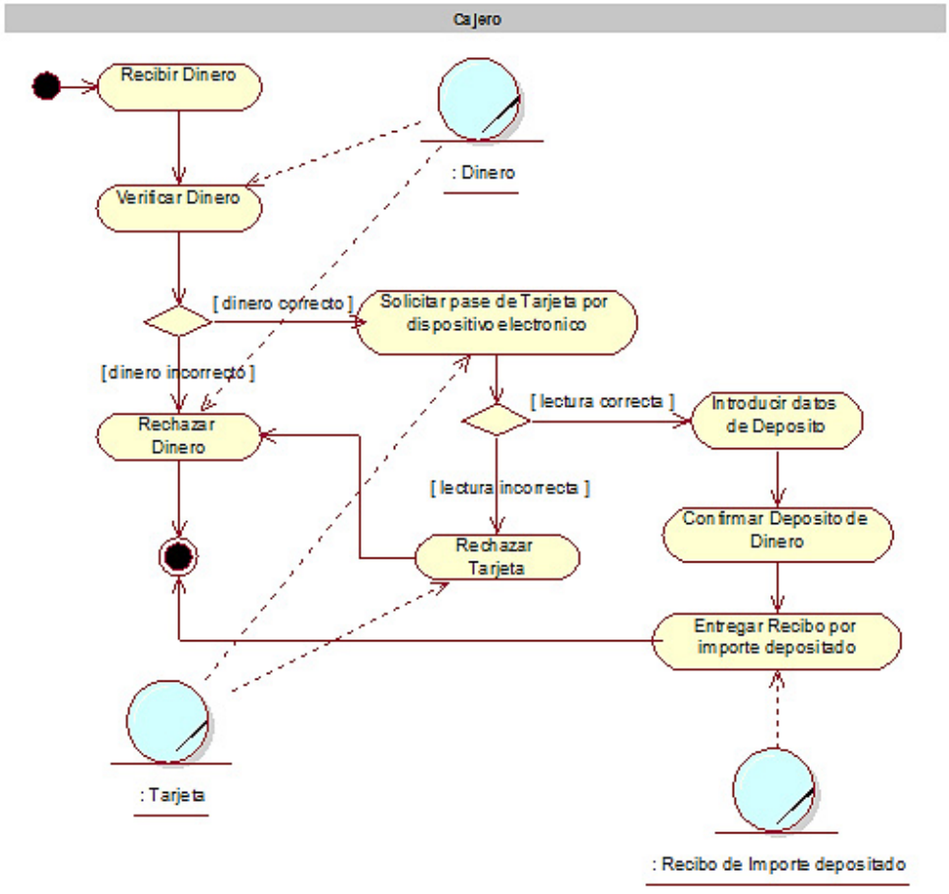


Figura N° 20: Ejemplo de Diagrama de actividades en UML
Fuente: http://modeladodesistemas1.blogspot.pe/

También existen notaciones para representar el flujo del proceso del negocio.

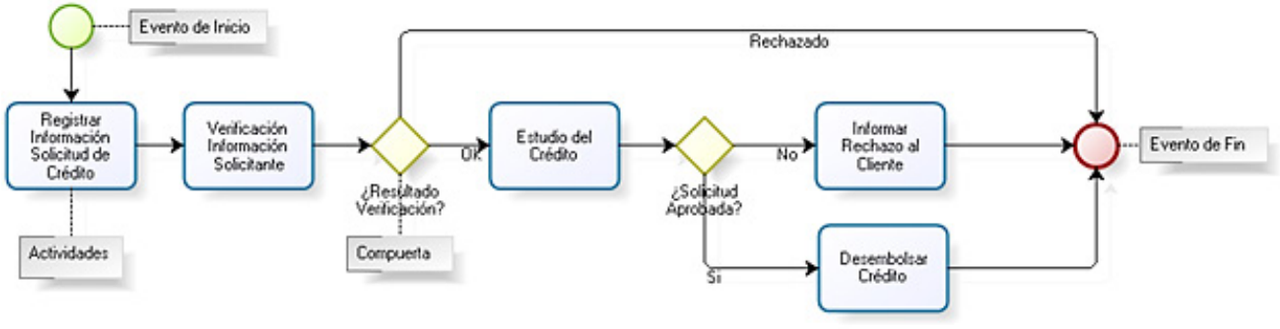


Figura N° 21: Ejemplo de Diagrama procesos de negocio
Fuente: https://javosantillan.wordpress.com/category/ingenieria-del-software/modelado-de-negocio-modelado-de-sistemas/

SÍNTESIS del TEMA II

REPRESENTACIÓN DEL ALGORITMO

- Diagrama formal de expresar a un algoritmo.
- El más conocido es el Pseudocódigo y el Diagrama de Flujo
- Diagrama de Flujo es usado en notaciones mas avanzadas para el desarrollo de software.

LECTURA SELECCIONADA N° 1

DISEÑO DE ALGORITMOS Y PROGRAMAS

Gallardo, J y García, C. Dpto. Lenguajes y Ciencias de la Computación – UMA. Ubicado en: http://www.lcc.uma.es/~pepeg/modula/temas/tema2.pdf

La principal razón por la que las personas aprenden a programar es para utilizar el ordenador como una herramienta para la resolución de problemas.

Ayudado por un ordenador, la obtención de la solución a un problema se puede dividir en dos fases:

1) Fase de resolución del problema.

2) Fase de implementación en el ordenador.

El resultado de la primera fase es el diseño de un algoritmo, que no es más que una secuencia ordenada de pasos que conduce a la solución de un problema concreto, sin ambigüedad alguna, en un tiempo finito. Sólo cuando dicho algoritmo haya sido probado y validado, se deberá entrar en detalles de implementación en un determinado lenguaje de programación; al algoritmo así expresado se le denomina programa.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como del ordenador que los ejecuta. El lenguaje de programación es tan sólo un medio para comunicarle al ordenador la secuencia de acciones a realizar y el ordenador sólo actúa como mecanismo para obtener la solución. En este sentido, podemos comparar la situación anterior con el hecho de explicar una receta a un cocinero en distintos idiomas. Mientras éste entienda el idioma, es indiferente el idioma elegido ya que el resultado final será siempre el mismo.

Todo algoritmo ha de cumplir necesariamente, las siguientes características:

- **Precisión:** el algoritmo debe indicar el orden de realización de cada acción, de forma clara y sin ambigüedades. Además, el algoritmo debe ser concreto en el sentido de contener sólo el número de pasos precisos para llegar a la solución (no deben darse pasos de más).
- **Repetitividad:** el algoritmo debe repetirse tantas veces como se quiera, alcanzándose siempre los mismos resultados para una misma entrada, independientemente del momento de ejecución.
- **Finitud:** el algoritmo debe terminar en algún momento.

Según esto, no toda secuencia ordenada de pasos a seguir puede considerarse como un algoritmo.

Por ejemplo, una receta para preparar un determinado plato no es un algoritmo, ya que:

- NO es repetible, pues para las mismas entradas (ingredientes) no se garantizarían los mismos resultados.
- NO es preciso, ya que en una receta concreta no se suelen especificar los grados de temperatura exactos, tipos de recipientes, calidad o tipo de ingredientes.

Además, a la hora de estudiar la calidad de un algoritmo, es deseable que los algoritmos presenten también otra serie de características como son:

- **Validez.** El algoritmo construido hace exactamente lo que se pretende hacer.
- **Eficiencia.** El algoritmo debe dar una solución en un tiempo razonable. Por ejemplo, para sumar 20 a un número dado podemos dar un algoritmo que sume uno veinte veces, pero esto no es muy eficiente. Sería mejor dar un algoritmo que lo haga de un modo más directo.
- **Optimización.** Se trata de dar respuesta a la cuestión de si el algoritmo diseñado para resolver el problema es el mejor. En este sentido y como norma general, será conveniente tener en cuenta que suele ser mejor un algoritmo sencillo que no uno complejo, siempre que el primero no sea extremadamente ineficiente.

En el algoritmo se plasman las tres partes fundamentales de una solución informática:

- **Entrada,** información dada al algoritmo.
- **Proceso,** cálculos necesarios para encontrar la solución del problema.
- **Salida,** resultados finales de los cálculos.

El algoritmo describe una transformación de los datos de entrada para obtener los datos de salida a través de un procesamiento de la información.

Los pasos a seguir en la fase de resolución del problema son tres:

1) Análisis del problema.

2) Diseño del algoritmo.

3) Verificación del algoritmo.

Análisis del problema

Es el primer paso para encontrar una solución computacional a un problema dado y requiere el máximo de creatividad por parte del programador.

El primer objetivo que nos debemos plantear es obtener una correcta comprensión de la naturaleza del problema. El análisis del problema exige una primera lectura del problema a fin de obtener una idea general de lo que se solicita. Una segunda lectura deberá servir para responder a las preguntas:

1) ¿Qué información debe proporcionar la resolución del problema?.

2) ¿Qué datos se necesitan para resolver el problema?.

La respuesta a la primera pregunta indicará los resultados deseados o salida del programa.

La respuesta a la segunda pregunta indicará qué datos se deben proporcionar o las entradas del problema.

Ejemplo: Las básculas calculan a partir del peso de un producto y de su precio por kilogramo, el correspondiente precio final y a partir de la cantidad de dinero entregada, la cantidad de dinero que debe ser devuelta.

Analizar el problema

La información de entrada para el problema es el peso y precio por kilogramo del producto junto con la cantidad de dinero entregada para pagar, a partir de lo cual se calculan las salidas del programa: el precio final del producto y la cantidad devuelta.

Diseño de algoritmos

Un ordenador no tiene la capacidad de pensar y resolver el problema por sí mismo; una vez que el problema ha quedado bien definido debemos plantearnos buscar una secuencia de pasos que lo resuelvan e indiquen al ordenador las instrucciones a ejecutar, es decir, hemos de encontrar un buen algoritmo.

Aunque en la solución de problemas sencillos parezca evidente la codificación en un lenguaje de programación concreto, es aconsejable el uso de algoritmos, a partir de los cuales se pasa al programa simplemente conociendo las reglas de sintaxis del lenguaje de programación a utilizar.

Sin embargo, la soluciones a problemas más complejos pueden requerir muchos más pasos. Las estrategias seguidas usualmente a la hora de encontrar algoritmos para problemas complejos son:

1) **Partición o divide y vencerás:** consiste en dividir un problema grande en unidades más pequeñas que puedan ser resueltas individualmente.

Ejemplo: Podemos dividir el problema de limpiar una casa en labores más simples correspondientes a limpiar cada habitación.

2) **Resolución por analogía:** Dado un problema, se trata de recordar algún problema similar que ya esté resuelto. Los dos problemas análogos pueden incluso pertenecer a áreas de conocimiento totalmente distintas.

Ejemplo: El cálculo de la media de las temperaturas de las provincias andaluzas y la media de las notas de los alumnos de una clase se realiza del mismo modo.

Evidentemente, la conjunción de ambas técnicas hace más efectiva la labor de programar: dividir un problema grande en trozos más pequeños ya resueltos.

Ejemplo: Consideremos el problema de calcular la longitud y la superficie de círculo dado su radio. Este problema se puede dividir en cuatro subproblemas:

1) Lectura, desde el teclado, de los datos necesarios.

2) Cálculo de la longitud.

3) Cálculo de la superficie.

4) Mostrar los resultados por pantalla.

El problema ha quedado reducido a cuatro subproblemas más simples. La solución a cada uno de estos subproblemas es un refinamiento del problema original.

El proceso para obtener la solución final consiste en descomponer en subproblemas más simples y, a continuación, dividir éstos nuevamente en otros subproblemas de menor complejidad, y así sucesivamente; la descomposición en subproblemas deberá continuar hasta que éstos se puedan resolver de forma directa. Este método de resolución de problemas no triviales da lugar a lo que se conoce como diseño descendente o diseño por refinamientos sucesivos:

Se comienza el proceso de solución con un enunciado muy general o abstracto de la solución del problema, de donde se identifican las tareas más importantes a ser realizadas, y el orden en el que se ejecutarán. A continuación, se procede repetidamente refinando por niveles, de manera que con cada descomposición sucesiva se obtiene una descripción más detallada, incluyendo nuevas acciones a realizar. El proceso finaliza cuando el algoritmo esté lo suficientemente detallado y completo para ser traducido a un lenguaje de programación.

Esta técnica es parte de las recomendaciones de una metodología general de desarrollo de programas denominada programación estructurada.

Verificación de algoritmos

Una vez que se ha descrito el algoritmo utilizando una herramienta adecuada, es necesario comprobar que realiza las tareas para las que fue diseñado y produce los resultados correctos y esperados a partir de la información de entrada.

Este proceso se conoce como prueba del algoritmo y consiste básicamente en recorrer todos los caminos posibles del algoritmo comprobando en cada caso que se obtienen los resultados esperados.

Para lo cual realizaremos una ejecución manual del algoritmo con datos significativos que abarquen todo el posible rango de valores y comprobaremos que la salida coincide con la esperada en cada caso. La aparición de errores puede conducir a tener que rediseñar determinadas partes del algoritmo que no funcionaban bien y a aplicar de nuevo el proceso de localización de errores, definiendo nuevos casos de prueba y recorriendo de nuevo el algoritmo con dichos datos.

Ejemplo: Diseñar un algoritmo que calcule el valor absoluto de un número.

Hay un único dato de entrada al algoritmo que es el número para el cual queremos calcular el valor absoluto. El dato de salida es el valor absoluto calculado para el número original.

Una primera aproximación al algoritmo podría ser:

1) Leer número del teclado

2) Valor absoluto ← – número

3) Escribir en pantalla valor absoluto

Si probamos el algoritmo con dato de entrada –3 obtenemos que el valor absoluto de éste es 3, lo cual es correcto. Pero el algoritmo diseñado no es correcto, ya que no funciona para datos de entrada positivos. En efecto, según el algoritmo anterior el valor absoluto de 3 es –3, lo cual no es cierto.

Dicha observación da lugar al siguiente algoritmo modificado, que funciona tanto para números positivos como negativos:

1) Leer número del teclado

2) Si número < 0 entonces

Valor absoluto ← – número en otro caso Valor absoluto ← número

3) Escribir en pantalla valor absoluto.

ACTIVIDAD FORMATIVA N° 2

Lee el tema de Representación de algoritmo. Complemente la información observando el vídeo Pseudocódigo y Diagrama de flujo | Iniciándose en la programación - 03 - Tutoriales y más, y los compara, elaborando una sencilla representación del desarrollo de programas, según la lectura N° 1.

INSTRUCCIONES:

1. Lee y analiza los contenidos del tema N° 2 y extrae los conceptos fundamentales para la representación del algoritmo.
2. Observa el vídeo **"Pseudocódigo y Diagrama de flujo | Iniciándose en la programación - 03 - Tutoriales y más"** y complementa la información de la representación de algoritmos.
3. Lee la lectura N° 1 y, dados los pasos de resolución de problemas, elabora un diagrama de flujo según los referidos pasos.

TEMA N° 3: PROGRAMACIÓN ESTRUCTURADA

¿Ya está listo para iniciarse en la programación? Estamos a punto de expresar los algoritmos en un lenguaje de programación: hemos conocido el algoritmo, sus características y sus partes, las instrucciones algorítmicas y su respectiva representación, pero aún nos falta definir el entorno de programación.

1. PROGRAMA

En realidad, el algoritmo que es escrito en un lenguaje de programación y puede ser ejecutado, constituye un programa. Entonces un programa sigue siendo un conjunto de tareas, pero esta vez de instrucciones (pasos) que serán ejecutados en una computadora.

[*] Triangulo.cpp

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      double b,h,area; //declaracion de variables
7
8      //ENTRADA
9      cout<<"Ingrese base: ";
10     cin>>b;
11
12     cout<<"Ingrese altura: ";
13     cin>>h;
14
15     //PROCESO
16     area = (b*h)/2;
17
18     //SALIDA
19     cout<<"El area del triangulo es: ";
20     cout<<area;
21     cout<<endl;   //fin de linea
22
23 return 0;
24 }
```

Figura N° 22: Ejemplo de Programa en lenguaje C/C++
Fuente: Carol Rojas M.

```java
package ejerciciosparte1;

import java.io.*;
public class Ejercicio1 {

    public static void main(String[] args) throws IOException{
            BufferedReader dato = new BufferedReader(new InputStreamReader(System.in));

            int num1,num2,num3;
            System.out.print("Ingrese numero1: ");
            num1 = Integer.parseInt(dato.readLine( ));
            System.out.print("Ingrese numero2: ");
            num2 = Integer.parseInt(dato.readLine( ));
            num3 = num1+num2;
            System.out.println("La suma es: " + num3 );
    }
}
```

Figura N° 23: Ejemplo de Programa en lenguaje Java
Fuente: Carol Rojas M.

2. LENGUAJE DE PROGRAMACIÓN

Conjunto de sentencias utilizadas para escribir secuencias de instrucciones para que ejecute un programa en una computadora.

El lenguaje de programación[4] sirve para escribir programas y permite la comunicación usuario (programador) versus máquina (PC).

Existen tres tipos de lenguaje de programación:

a. Lenguaje máquina: Lenguaje de programación que la computadora interpreta y ejecuta directamente, y está compuesto de instrucciones codificadas en binario (0 , 1).

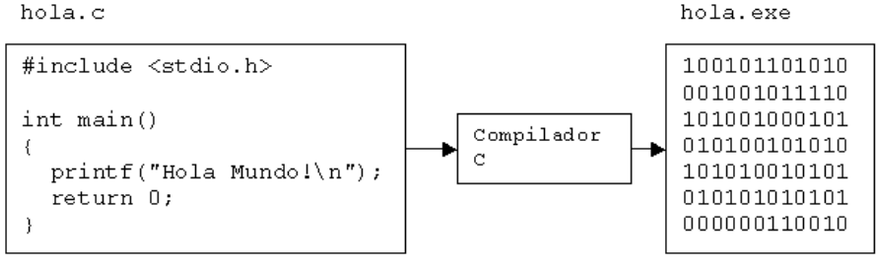


Figura N° 24: Ejemplo de Lenguaje máquina
Fuente: http://go.yuri.at/juegos/clase1/clase1_introduccion.html

b. Lenguaje de bajo nivel: También llamados lenguajes ensambladores, permiten al programador escribir instrucciones de un programa usando abreviaturas del lenguaje natural (inglés), también llamadas palabras nemotécnicas (ADD, DIV, SUB, etc).

[4] Marcelo Villalobos, Ricardo. (2014). Fundamentos de Programación C++. Perú. Editorial Macro.

```
MODEL   Small
Stack               100h
DATASEG
Mensaje             DB 'Hola, Mundo',13,10,'$'
CODESEG
Inicio:             mov ax,@data
                    mov ds,ax                   ; DS ahora apunta al segmento de datos.
                    mov ah,9                    ; Función del DOS para impresión de cadenas.
                    mov dx,OFFSET Mensaje       ; Apuntador al mensaje 'Hola Mundo'.
                    int 21h
                    mov ah,4Ch                  ; Transferencia del control al DOS.
                    int 21h
END                 Inicio
```

Figura N° 25: Ejemplo de Lenguaje de bajo nivel
Fuente: https://arquitecturacomputadoreshoy.wordpress.com/2014/10/page/2/

c. Lenguaje de alto nivel: Permite al programador escribir las instrucciones de un programa, utilizando palabras o expresiones sintácticas muy similares al lenguaje natural (inglés).

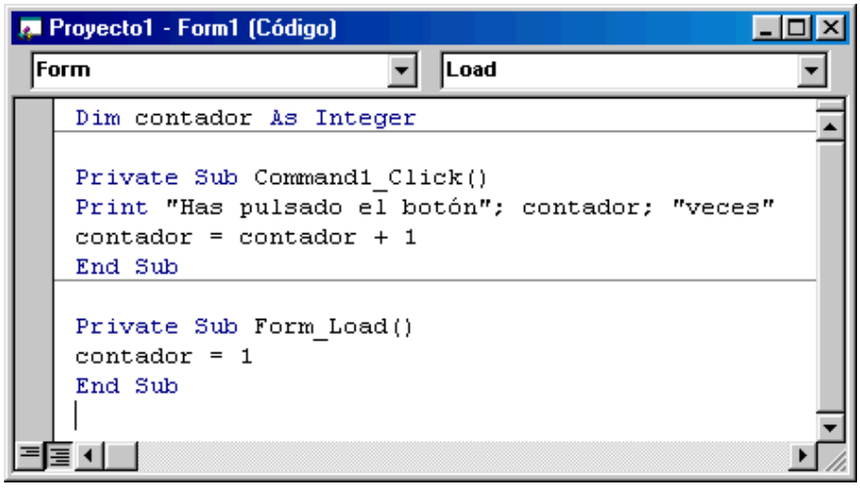
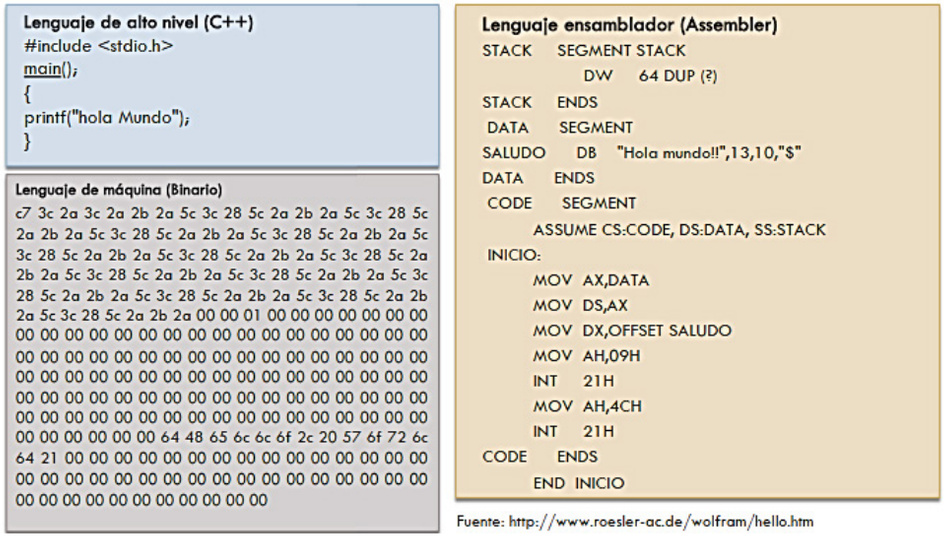


Figura N° 26: Ejemplo de Lenguaje de alto nivel
Fuente: http://www.adrformacion.com/curso/visualbasic/leccion2/ElementosLenguaje.htm

A continuación, se muestra un ejemplo en los tres tipos de lenguaje de programación descritos.



Fuente: http://www.roesler-ac.de/wolfram/hello.htm

Figura N° 27: Ejemplo de Lenguajes de programación
Fuente: http://slideplayer.es/slide/1075615/

3. PROGRAMAS TRADUCTORES

Para elaborar un programa es necesario conocer cómo es que lo que escribimos en cualquier entorno de programación es traducido hacia la máquina.

Los traductores se dividen en compiladores e intérpretes: [5]

3.1 INTÉRPRETE:

Es un traductor que toma un programa fuente, traduce una sentencia y, a continuación, lo ejecuta y realiza el mismo procedimiento para las demás sentencias.

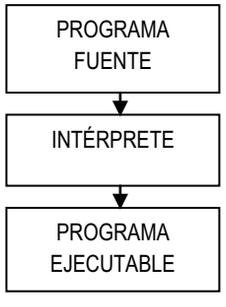


Figura N° 28: Representación del intérprete
Fuente: Joyanes, L. (2008). Fundamentos de programación

3.2 COMPILADOR:

Es un traductor que toma un programa fuente, y lo traduce en un solo bloque. Para ello, crea un programa objeto,

[5] Joyanes, L. (2008). Fundamentos de programación. España:. McGRAW-HILL.

el cual tiene el código traducido y luego es invocado por el programa enlazador para su ejecución.

Figura N° 29: Representación del compilador
Fuente: Joyanes, L. (2008). Fundamentos de programación

4. DEFINICIÓN DE PROGRAMACIÓN ESTRUCTURADA.

Existen dos enfoques de programación y, por lo tanto, de desarrollo de software, muy utilizados en la actualidad, el Enfoque estructurado y el Enfoque orientado a objetos, cada uno con sus conceptos, técnicas y herramientas.

El Enfoque orientado a objetos emplea algunas técnicas de programación del Enfoque estructurado, pero tiene como principal concepto al Objeto con atributos y comportamientos comunes y a la Clase (colección de objetos)

Figura N° 30: Ejemplo de clases y cbjetos
Fuente: http://es.slideshare.net/cpavella/6-clases-objetos

Estas clases pueden heredar los atributos y comportamientos a otras subclases:

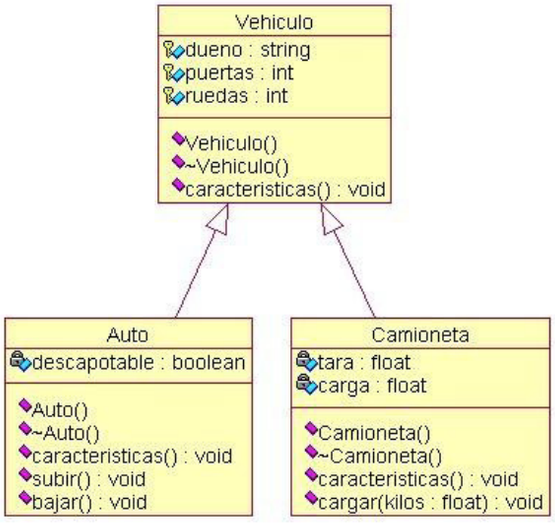


Figura N° 31: Ejemplo de herencia de clases
Fuente: http://users.dcc.uchile.cl/~psalinas/uml/modelo.html

En esta asignatura, por ser de formación básica en su carrera y para facilitar el inicio del estudiante en la elaboración de programas, usaremos la Programación del Enfoque estructurado.

La Programación estructurada es un conjunto de técnicas para escribir, verificar, depurar y mantener los programas, realizando refinamientos sucesivos, es decir, un todo se divide en segmentos más sencillos o de menor complejidad (diseño descendente) que al darles solución, se proceden a unificar.

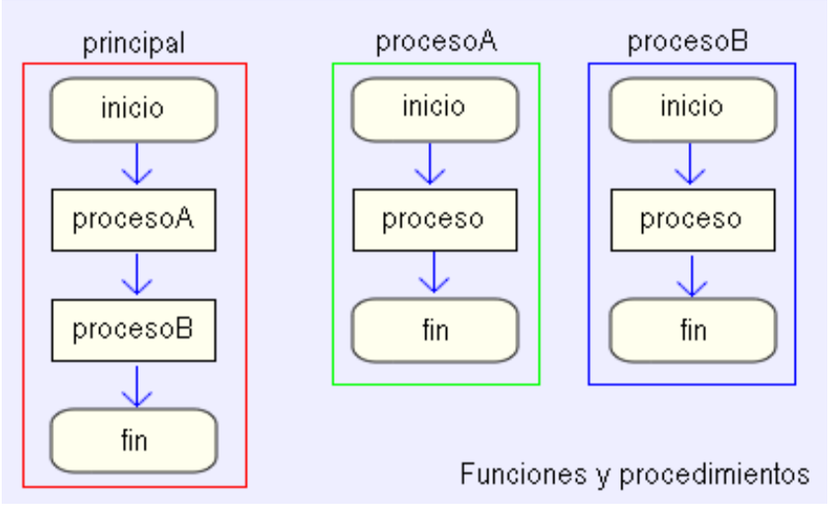


Figura N° 32: Ejemplo de Enfoque estructurado
Fuente: http://programacioncitlajessiyaz.blogspot.pe/p/unidad-5-funciones.html

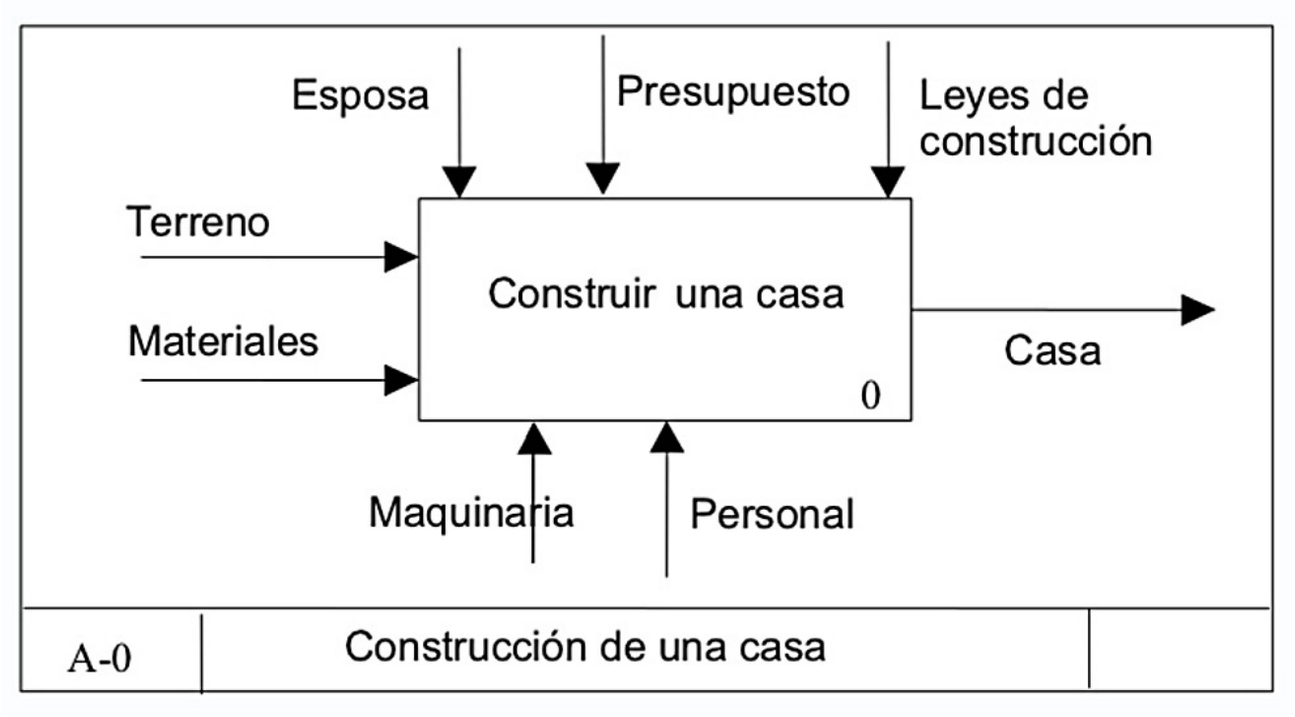


Figura N° 33: Ejemplo de Enfoque estructurado con IDEF
Fuente: http://es.slideshare.net/JuanPablo157/diagramas-idef-0-y-3

Estructura de un programa en C/C++

Para iniciar la elaboración de programas en esta asignatura, se muestra la Estructura de un programa simple en C++, en el tema de Módulos de programa, la estructura del programa tendrá algunas modificaciones.

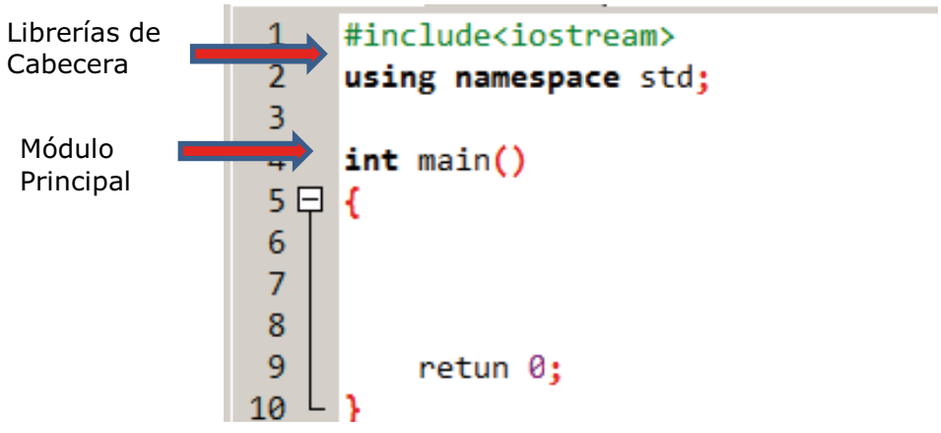


Figura N° 34: Estructura básica de un programa en C/C++
Fuente: Carol Rojas M.

Descripción:

- El símbolo # es una directiva del procesador que permite acceder la carpeta include.

- La librería include es propia del compilador, donde se encuentran las librerías de cabecera (extensión .h) del lenguaje C/C++
- Una de las librerías de cabecera es iostream, (i:imput, o:output, stream: flujo de cadenas) que permite reconocer las instrucciones de entrada, salida del programa.
- using namespace std; uso del espacio de conjunto de símbolos estándar.
- El módulo principal main( ), en el cual se escribirán las instrucciones de programa y las invocaciones a otros módulos de programa.

En otros lenguajes de programación como, por ejemplo, en java, se presenta la siguiente estructura.

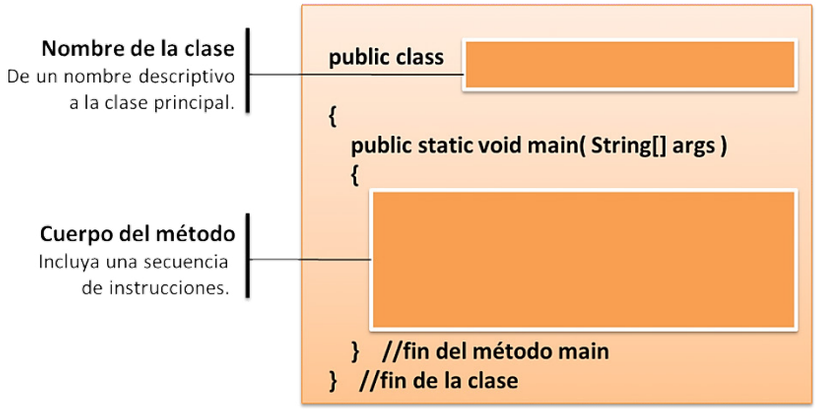


Figura N° 35: Estructura básica de un programa en Java
Fuente: http://profejavaoramas.blogspot.pe/2010/04/estructura-de-un-programa-en-java.html

SÍNTESIS del TEMA III

PROGRAMACIÓN ESTRUCTURADA

Conjunto de técnicas para elaborar y depurar programas.

Requiere de un programa traductor, para convertir a código máquina lo escrito en el lenguaje de programación.

Traductores:

- Intérprete
- Compilador

ACTIVIDAD FORMATIVA Nº 3

Identifica los elementos de representación de los diagramas de flujo y sus características y elabora un diagrama para los casos propuestos.

INSTRUCCIONES:

1. Dado los siguientes procesos secuenciales:

a. El proceso de calcular su promedio final, ingresando las notas de C1, EP, C2 y EF, dada la fórmula: PF = C1 (20%) + EP (20%) + C2 (20%) + EF (40%).

b. El proceso de calcular un valor con la regla de tres simple.

Elaborar el Diagrama de flujo de cada algoritmo (conjunto de acciones), usando la tabla que identifica entrada, salida, y proceso.

ENTRADA DATOS A USAR	PROCESO (PROCEDIMIENTO)	SALIDA DATOS PARA REPORTAR

El instrumento para calificar es una Lista de cotejo con los siguientes criterios: para un archivo Word con la tabla de partes del algoritmo y el archivo del diagrama en PseInt en una carpeta ApellidoNombreAlumno.zip.

CRITERIO	PUNTAJE
Elaboración de la tabla con las tres partes del algoritmo.	3
Uso del programa PseInt, para la elaboración del diagrama de flujo.	3
Dar nombre al proceso requerido.	2
Usar el símbolo de entrada de datos.	3
Usar el símbolo de salida de datos.	3
Usar el símbolo de proceso de datos.	3
Mostrar la venta de resultados del proceso.	3

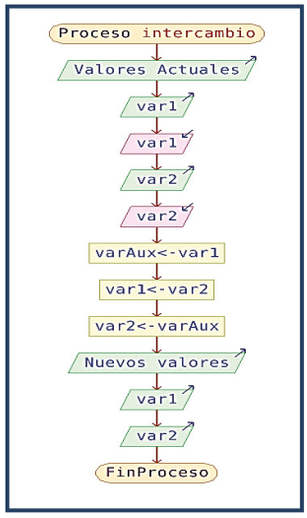
2. Se recomienda utilizar el software en línea: Programa PSeInt http://pseint.sourceforge.net/index.php?page=descargas.php.

3. Complementa tu información observando el vídeo: "**Diagrama de flujo con PSeint".**

Ejemplos:

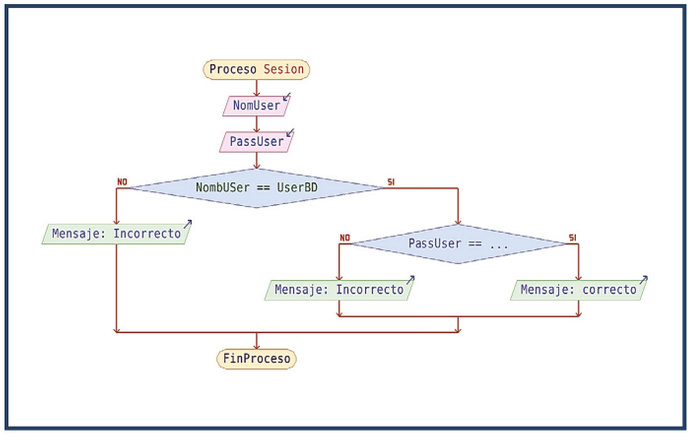
- El proceso de intercambiar dos valores numéricos, cada uno en su respectiva variable (Var1 y Var2).

ENTRADA Datos a usar	PROCESO (Procedimiento)	SALIDA Datos para reportar
Var1 Var2	VarAux = Var1 Var1 = Var2 Var2 = VarAux	"El valor 1 es: ", Var1 "El valor 2 es: ", Var2



- El proceso de Iniciar Sesión en una aplicación.

ENTRADA Datos a usar	PROCESO (Procedimiento)	SALIDA Datos para reportar
NombreUser PasswordUser	Si (NombreUser == UserBD) Entonces Si (PasswordUser == PassBD) Entonces Mensaje = "Correcto" Sino Mensaje = "Password Incorrecto" Sino Mensaje = "Password Incorrecto"	Mensaje



GLOSARIO DE LA UNIDAD I

C

CONSTANTE
En un programa, es un valor que no puede ser modificado para la ejecución.

E

ENTORNO
Conjunto de datos, personas, actividades, restricciones que definen una situación en particular.

I

INSTRUCCION
En programación, es una orden, operación, acción que se escribe en un programa y es ejecutado por el computador.

L

LENGUAJE DE PROGRAMACIÓN
Lenguaje artifical con un conjunto de símbolos, sintaxis y semántica para elaborar programas.

O

OPERADOR
Es un símbolo que se escribe en una instrucción, y permite ejecutar sentencias lógicas, aritméticas, relacionales y de asignación.

P

PROCESO
Conjunto de actividades (pasos o tareas) realizadas por una persona, con o sin ayuda de algun equipo, en una organización.

PROGRAMA FUENTE
Algoritmo codificado en un lenguaje de programación que administra otros módulos de programa, y es necesario para la ejecución del programa.

T

TIPO DE DATO
Es un atributo o característica de un dato, es decir, es la especificación de un rango de valores (enteros, reales, cadenas) en la memoria del computador.

BIBLIOGRAFÍA DE LA UNIDAD I

Joyanes, L. (2008). *Fundamentos de programación*. Madrid: McGRAW-HILL.

Marcelo, R. (2014). *Fundamentos de programación C++*. Lima: Editorial Macro.

Suarez, G. (2003). *Algoritmos y diagramas de flujo*. Lima: Libros y publicaciones.

AUTOEVALUACIÓN DE LA UNIDAD I

1. Indique la alternativa que describa la característica **"Ser definido"** de un algoritmo:

   A) Las tareas de un algoritmo tienen un orden de realización.

   B) Las tareas garantizan quesiempre se obtendrá el mismo resultado.

   C) Las tareas desarrolladas deben permitir ponerlas a prueba antes de ejecutar.

   D) Se emplean los recursos necesarios para el desarrollo de las tareas.

   E) Se debe expresar en una forma estándar, comúnmente aceptada.

2. La característica "Eficiencia" de un algoritmo, se entiende como:

   A) No debe tener ningún error en la validación.

   B) Se hace uso de todos los recursos, aún no necesarios.

   C) Tiene un inicio y un fin de tareas a desarrollar.

   D) Uso adecuado de recursos, solo los requeridos.

   E) Es independiente al lenguaje de programación a utilizar.

3. Las partes de un algoritmo son:

   A) Entrada, Proceso, Diagrama.

   B) Entrada, Diagrama, Salida.

   C) Entrada, Proceso, Programa.

   D) Programa, Proceso, Salida.

   E) Entrada, Proceso, Salida.

4. En la elaboración de un algoritmo, se define a la variable como:

   A) Todo el algoritmo a desarrollar.

   B) Incógnita en la salida de datos.

   C) Espacio de memoria en el computador.

   D) Una fórmula matemática.

   E) Las actividades del algoritmo.

5. En la programación estructurada, se presentan los siguientes diagramas de flujo:

A) Pseudocódigo, Diagrama de actividades, Diagrama N-S.

B) Diagrama de actividades, Diagrama de flujo, Diagrama N-S.

C) Pseudocódigo, Diagrama de flujo, Diagrama N-S.

D) Diagrama de procesos de negocio, Diagrama de actividades, Diagrama N-S.

E) Pseudocódigo, Diagrama de procesos de negocio, Diagrama N-S.

6. Los tipos de lenguaje de programación conocidos son:

A) Lenguaje máquina, Programa traductor, Lenguaje alto nivel.

B) Lenguaje bajo nivel, Programa traductor, Lenguaje alto nivel.

C) Compilador, Programa traductor, Lenguaje alto nivel.

D) Intérprete, Lenguaje bajo nivel, Lenguaje alto nivel.

E) Lenguaje máquina, Lenguaje bajo nivel, Lenguaje alto nivel.

7. El Lenguaje de bajo nivel, tiene como instrucciones en el programa:

A) Los símbolos de cero y uno.

B) Los símbolos en lenguaje natural.

C) Los símbolos nemotécnicos.

D) Los símbolos de Diagrama N-S.

E) Los símbolos en Diagrama de Flujo.

8. El programa traductor denominado compilador realiza el siguiente flujo de trabajo.

A) Obtiene el programa fuente, genera el código objeto, lo enlaza al código máquina y genera el programa ejecutable.

B) Obtiene el programa fuente, genera el código algoritmo, lo enlaza al código máquina y genera el programa ejecutable.

C) Obtiene el programa fuente, genera el código objeto, lo enlaza al código fuente y genera el programa ejecutable.

D) Obtiene el programa fuente, genera el código máquina, lo enlaza al código fuente y genera el programa ejecutable.

E) Obtiene el programa fuente, genera el código algoritmo, lo enlaza al código fuente y genera el programa ejecutable.

9. La programación estructurada, a diferencia de la programación orientada a objetos, se fundamenta en:

   A) Dividir en segmentos más sencillos o de menor complejidad, para unificar en la solución.

   B) Mantener el problema como un todo y ejecutarlo en un solo programa bloque de solución.

   C) Realizar la herencia entre clases con objetos de atributos y comportamiento similares, como parte de la solución.

   D) Dividir el programa en clases y objetos, para luego unificar en la solución.

   E) Realizar la herencia entre segmentos de menor complejidad, para unificar en al solución.

10. En la estructura de un programa en C++, el módulo principal realiza:

   A) Exclusivamente la entrada de datos.

   B) El acceso a la capeta include.

   C) El acceso al uso de símbolos estándar.

   D) Las invocaciones a otros módulos.

   E) Exclusivamente el proceso del algoritmo.

UNIDAD II

ESTRUCTURAS DE CONTROL PARA LA PROGRAMACIÓN: SELECTIVAS.

DIAGRAMA DE PRESENTACIÓN DE LA UNIDAD II

Al finalizar la unidad, el estudiante será capaz de elaborar programas computacionales, aplicando las estructuras de control, secuencial y selectivas.

CONTENIDOS	ACTIVIDADES FORMATIVAS (HABILIDADES Y ACTITUDES)	SISTEMA DE EVALUACIÓN (TÉCNICAS Y CRITERIOS)
Tema N° 1 : Estructuras de control para la programación: 1 Definición 2 Estructura de control secuencial. **Tema N° 2 : Estructura de control selectiva simple y selectiva compuesta:** 1 Definición de Estructura de control selectiva simple. 2 Definición de Estructura de control selectiva compuesta. **Tema N° 3 : Estructura de control múltiple.** 1 Definición de Estructura de control múltiple. 2 Ejemplo de Estructuras. 3 Electiva múltiple.	• Lee y Analiza el tema Estructuras de Control para la programación, Secuencial. Observa el video La estructura secuencial - Parte I / C++ y elabora un programa en lenguaje C/C++ por cada situación propuesta. • Lee y Analiza el tema Estructuras de Control para la programación, Secuencial. Observa el video La estructura condicional simple/ compuesta (if) / C++ y elabora un programa en lenguaje C/C++ por cada situación propuesta complementando con los conceptos según la lectura N°1. • Lee y Analiza el tema Estructuras de Control para la programación, Secuencial. Observa el video La estructura de selección múltiple (Switch) / C++ y elabora un programa en lenguaje C/C++ por cada situación propuesta.	**Procedimientos e indicadores de evaluación permanente:** • Entrega puntual de trabajos realizados. • Calidad, coherencia y pertinencia de contenidos desarrollados. • Prueba teórico-práctica, individual. • Actividades desarrolladas en sesiones tutorizadas. **Criterios de evaluación para Diagrama de representación:** • Cuadro de identificación de partes del algoritmo: Entrada, Proceso, salida. • Programas elaborados con estructuras selectivas en un lenguaje de programación propuesto, considerando las partes del algoritmo.

RECURSOS:

VIDEOS:

Tema Nº 1: La estructura secuencial - Parte I / C++.
https://www.youtube.com/watch?v=7SYjHAXvB6Q

Tema Nº 2: La estructura condicional simple/compuesta (if) / C++.
https://www.youtube.com/watch?v=Hs4yRc3K9I8

Tema Nº 3: La estructura de selección múltiple (Switch) / C++.
https://www.youtube.com/watch?v=7c3RSBz7si0

DIAPOSITIVAS ELABORADAS POR EL DOCENTE:

Lectura complementaria:

Lectura Seleccionada Nº 1

Estructuras de control: condicionales. Departamento de Ecuaciones Diferenciales y Análisis Numérico. Universidad de Sevilla. Ubicada en:
http://departamento.us.es/edan/php/asig/LICFIS/LFIPC/Tema5FISPC0809.pdf

INSTRUMENTO DE EVALUACIÓN	Prueba mixta N° 1
BIBLIOGRAFÍA (BÁSICA Y COMPLEMENTARIA)	**BASICA** Joyanes, L. (2008) *Fundamentos de programación.* Cuarta Edición. España : McGRAW-HILL. **COMPLEMENTARIA** Carrasco, A (2005). *Principios de programación.* Algoritmos y su creación en C++. Perú: AC Editores.
RECURSOS EDUCATIVOS DIGITALES	Numérico, D. d. (2009). Universidad de Sevilla. Ubicado en: http://departamento.us.es/edan/php/asig/LICFIS/LFIPC/Tema5FISPC0809.pdf.

TEMA N° 1:
ESTRUCTURAS DE CONTROL PARA LA PROGRAMACIÓN

Estimado estudiante, para esta segunda unidad, se presentan las estructuras de control o sentencias básicas para la programación, las cuales le permitirán elaborar programas en diferentes lenguajes, a través de ejemplos en situaciones de la realidad que usted conoce.

1. DEFINICIÓN

Conjunto de técnicas que permite elaborar programas en cualquier lenguaje de programación.

Las estructuras de control o sentencias básicas son una de estas técnicas, y permiten iniciarse sin dificultad en la programación. Se conocen como:

- Secuenciales
- Selectivas
- Repetitivas

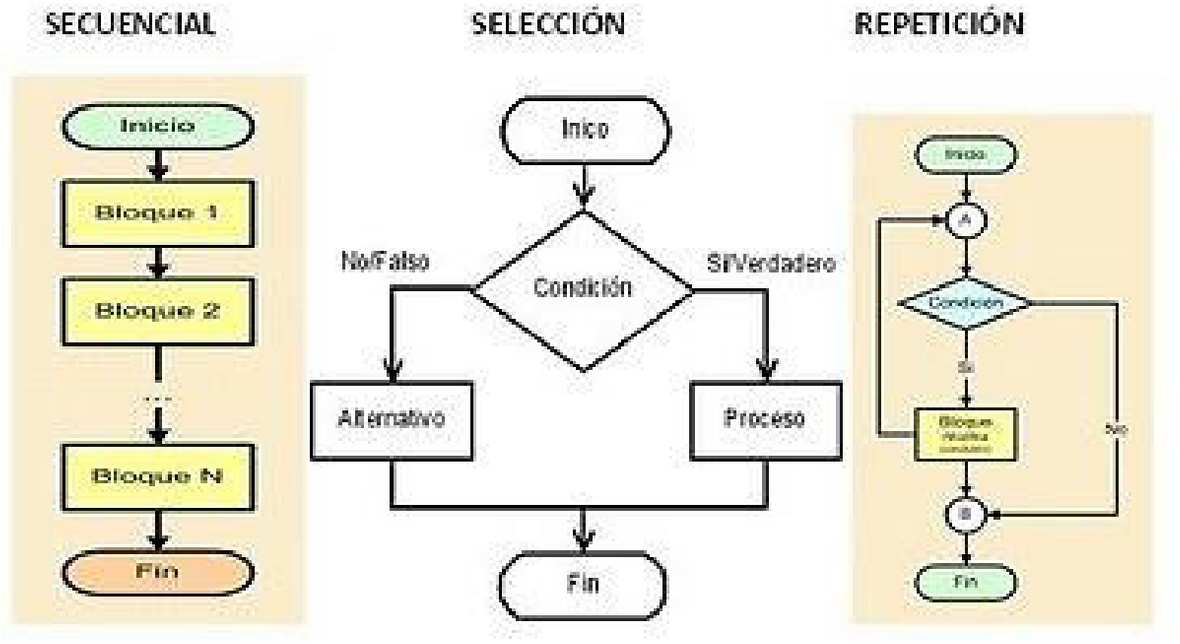


Figura N° 36: Ejemplos de Estructuras de control para la programación
Fuente: http://informatica.iesvalledeljerteplasencia.es/wordpress/diseno-de-programas-pseudocodigo-y-diagramas/

2. ESTRUCTURA DE CONTROL SECUENCIAL[6]

Es una estructura paso a paso, sin bifurcaciones ni repeticiones, donde la salida de un proceso es entrada para otro.

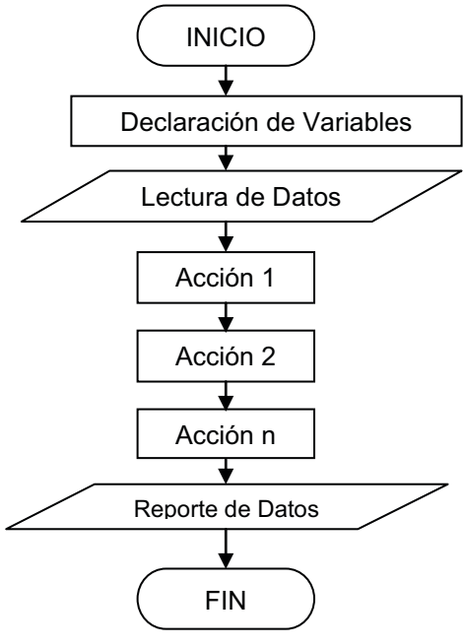


Figura N° 37: Diagrama de flujo para una Estructura de control secuencial
Fuente: Carol Rojas M.

Ejemplo de Estructura secuencial:

Tenemos el requerimiento de calcular el área de un triángulo, conociendo la fórmula:

Área = (base * altura) / 2

Solución:

Se sugiere definir las variables para el proceso de entrada, asignándoles un nombre que se aproxime a su significado, según el contexto que se trabaje:

Definición de variables:

Base : valor de la base del triángulo.

Altura: valor de la altura del triangulo.

Área: valor del resultado del cálculo del área del triángulo.

Esta definición de variables será usada para el ingreso de valores y para la recepción del valor calculado respectivamente, luego de haberle asignado un tipo de dato, según el lenguaje de programación que esté utilizando.

[6] Carrasco, A. (2005). Principios de programación. Algoritmos y su creación en C++. Perú: AC Editores.

Recuerde que algunos lenguajes de programación diferencian entre mayúsculas y minúsculas, y que no es muy recomendable usar nombre de variables que no guarden relación con su significado, ya que no podrá ser de fácil corrección o mantenimiento, a un futuro por otros desarrolladores de programas o software.

Solución en Diagrama de flujo:

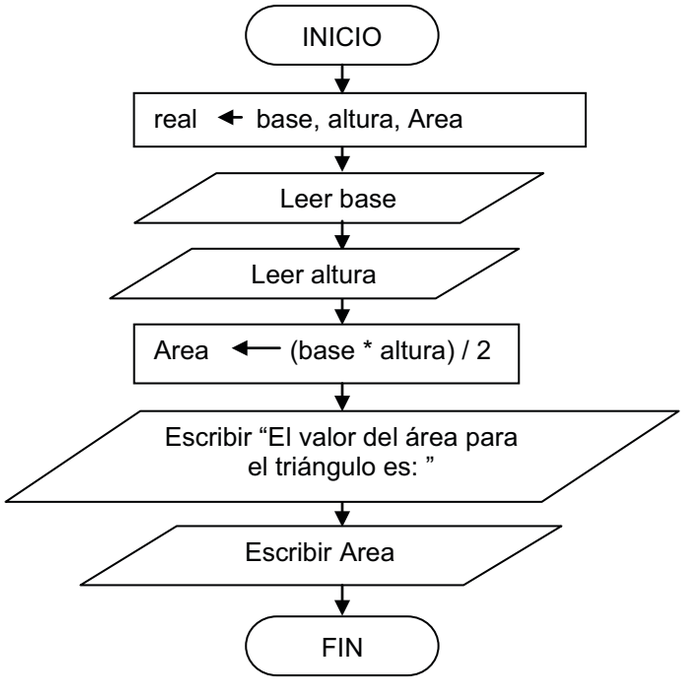


Figura N° 38: Solución de Diagrama de flujo para Ejemplo 1 de una Estructura de control secuencial
Fuente: Carol Rojas M.

Solución en Código C/C++:

```
#include<iostream>
using namespace std;

int main()
{
    double base, altura, Area;

    //ENTRADA DE DATOS
    cout<<"Ingrese base: ";
    cin>>base;

    cout<<"Ingrese altura: ";
    cin>>altura;

    //PROCESO: cálculo
    Area = (base * altura)/2;

    //SALIDA DE DATOS
    cout<<"El area es: ";
    cout<<Area;

    return 0;
}
```

Figura N° 39: Solución en Código C/C++ para Ejemplo 1 de una Estructura de control secuencial
Fuente: Carol Rojas M.

Otro Ejemplo de estructura secuencial:

Tenemos los datos de presión, el volumen y la temperatura de una masa de aire se relacionan por la formula:

Masa = (presión * volumen) / (0.37 * (temperatura + 460))

Solución:

De igual forma que el ejemplo anterior, se sugiere definir las variables asignándoles un nombre que se aproxime a su significado, según el contexto que se trabaje:

Definición de variables:

- **Presión** : valor de la presión a ingresar.
- **Volumen** : valor del volumen a ingresar.
- **Temperatura** : valor de la temperatura ingresar.
- **Masa** : valor del resultado del cálculo.

Solución en Diagrama de flujo:

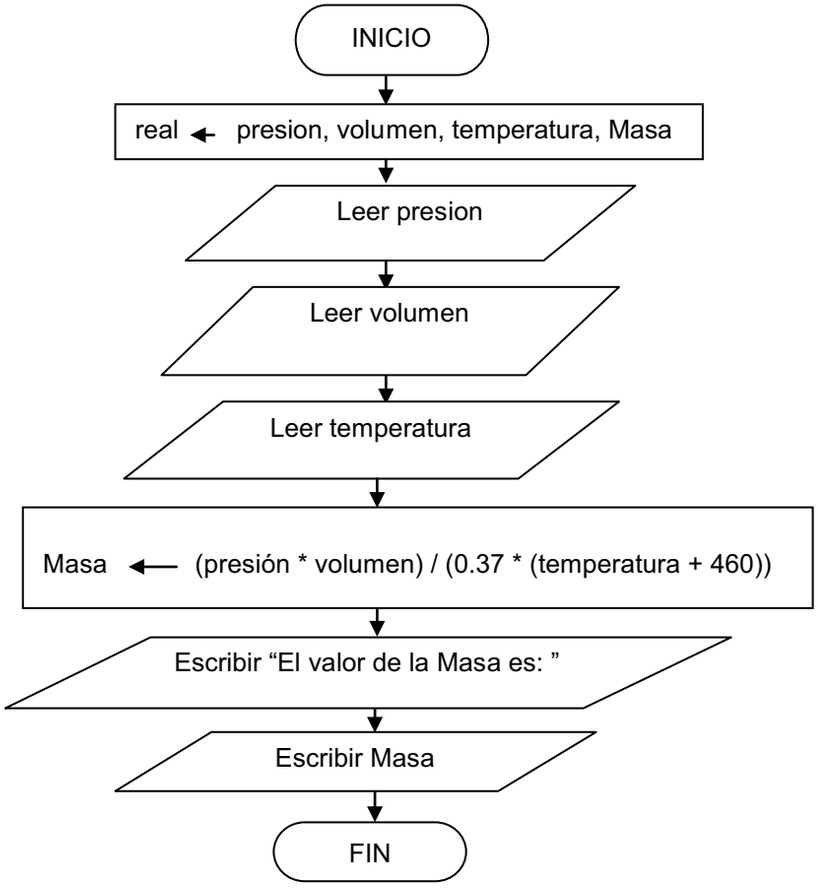


Figura N° 40: Solución de Diagrama de flujo para Ejemplo2 de una Estructura de Control Secuencial
Fuente: Carol Rojas M.

Solución en Código C/C++:

```
#include<iostream>
using namespace std;

int main()
{
    double presion, volumen, temperatura, Masa;

    //ENTRADA DE DATOS
    cout<<"Ingrese presion: ";
    cin>>presion;

    cout<<"Ingrese volumen: ";
    cin>>volumen;

    cout<<"Ingrese temperatura: ";
    cin>>temperatura;

    //PROCESO: cálculo
    Masa = (presion * volumen) / (0.37 * (temperatura + 460));

    //SALIDA DE DATOS
    cout<<"El valor de Masa es: ";
    cout<<Masa;

    return 0;
}
```

Figura N° 41: Solución en Código C/C++ para Ejemplo 2 de una Estructura de control secuencial
Fuente: Carol Rojas M.

Un ejemplo más de Estructura secuencial:

Ingresar el sueldo de tres empleados y aplicarles un incremento de 10%, 20%, 30% respectivamente. Reportar los nuevos valores de sueldo.

Solución:

También se debe definir variables pero, en esta oportunidad, se propondrá un cuadro que le permita definir las tres partes del algoritmo de solución:

Cuadro N° 5: Solución con partes del algoritmo para Ejemplo 3 de una Estructura de control secuencial

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR
SueldoEmp1 SueldoEmp2 SueldoEmp3 IncrementoEmp1 IncrementoEmp2 IncrementoEmp3 NuevoSueldoEmp1 NuevoSueldoEmp2 NuevoSueldoEmp3	SueldoEmp1 SueldoEmp2 SueldoEmp3	IncrementoEmp1 = SueldoEmp1 * 0.1; IncrementoEmp2 = SueldoEmp1 * 0.2; IncrementoEmp3 = SueldoEmp3 * 0.3; NuevoSueldoEmp1 = SueldoEmp1 + IncrementoEmp1; NuevoSueldoEmp2 = SueldoEmp2 + IncrementoEmp2; NuevoSueldoEmp3 = SueldoEmp3 + IncrementoEmp3;	NuevoSueldoEmp1 NuevoSueldoEmp2 NuevoSueldoEmp3

Fuente: Carol Rojas M.

Solución en Diagrama de flujo:

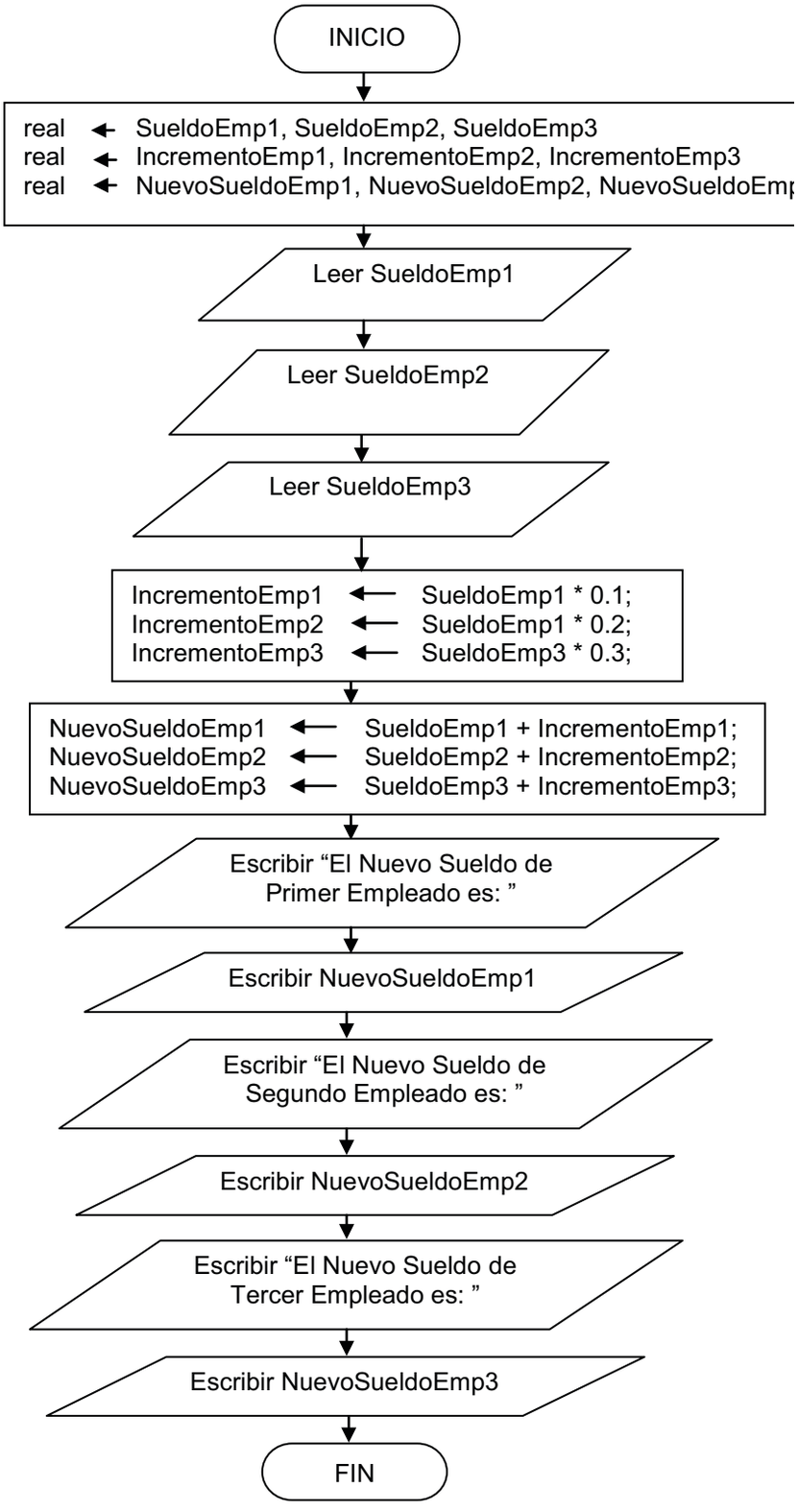


Figura N° 42: Solución de Diagrama de flujo para Ejemplo3 de una Estructura de control secuencial
Fuente: Carol Rojas M.

Solución en Código C/C++:

```cpp
#include<iostream>
using namespace std;

int main()
{
    double SueldoEmp1, SueldoEmp2, SueldoEmp3;
    double IncrementoEmp1, IncrementoEmp2, IncrementoEmp3;
    double NuevoSueldoEmp1, NuevoSueldoEmp2, NuevoSueldoEmp3;

    //ENTRADA DE DATOS

    cout<<"Ingrese Sueldo de Primer Empleado: ";
    cin>>SueldoEmp1;

    cout<<"Ingrese Sueldo de Segundo Empleado: ";
    cin>>SueldoEmp2;

    cout<<"Ingrese Sueldo de Tercer Empleado: ";
    cin>>SueldoEmp3;

    //PROCESO:
    //a. cálculo de incremento según porcentaje

    IncrementoEmp1 = SueldoEmp1 * 0.1;
    IncrementoEmp2 = SueldoEmp1 * 0.2;
    IncrementoEmp3 = SueldoEmp3 * 0.3;

    //b. cálculo de nuevo sueldo según incremento

    NuevoSueldoEmp1 = SueldoEmp1 + IncrementoEmp1;
    NuevoSueldoEmp2 = SueldoEmp2 + IncrementoEmp2;
    NuevoSueldoEmp3 = SueldoEmp3 + IncrementoEmp3;

    //SALIDA DE DATOS
    cout<<endl;
    cout<<"El Nuevo Sueldo de Primer Empleado es: ";
    cout<<NuevoSueldoEmp1<<endl;

    cout<<"El Nuevo Sueldo de Segundo Empleado es: ";
    cout<<NuevoSueldoEmp2<<endl;

    cout<<"El Nuevo Sueldo de Tercer Empleado es: ";
    cout<<NuevoSueldoEmp3<<endl;

    return 0;
}
```

Figura N° 43: Solución en Código C/C++ para Ejemplo 3 de una Estructura de control secuencial
Fuente: Carol Rojas M.

SINTESIS del TEMA I

PROGRAMACIÓN ESTRUCTURADA

Técnicas para programar:

Secuencial
Selectiva
Repetitiva

Estructura de Control Secuencial:

Es lineal, no tiene condiciones ni repeticiones.

ACTIVIDAD FORMATIVA Nº 1

Elabora el programa de cómputo para cada situación propuesta

INSTRUCCIONES:

1. Lee y analiza el tema N° 1 y extrae las ideas fundamentales de la estructura de control secuencial para la programación.

2. Observe el vpídeo **"La estructura secuencial - Parte I / C++"** y complemente la información obtenida en el tema N° 1.

3. Elabore un programa en lenguaje C/C++ para las siguientes situaciones:

3.1. En un hospital existen tres áreas: Ginecología, Pediatría y Traumatología. El presupuesto anual del hospital se reparte conforme con la siguiente tabla:

- Área Porcentaje del presupuesto
- Ginecología 40%
- Traumatología 30%
- Pediatría 30%

Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestal.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

3.2. Calcular el monto total a pagar, de acuerdo con el procedimiento:

a. Ingrese el nombre de un producto.

b. Ingrese el precio del producto.

c. Ingrese la cantidad de producto a vender.

d. Calcule el monto bruto.

e. Ingrese el porcentaje de descuento 20%

f. Calcule el monto de descuento.

g. Calcule el monto total de pago

h. Reportar: monto bruto, monto de descuento, monto total de pago.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

TEMA N° 2:
ESTRUCTURA DE CONTROL SELECTIVA: SIMPLE Y SELECTIVA COMPUESTA

Como se habrá dado cuenta, no siempre se tendrá casos o procesos en que la solución sea desarrollada de forma secuencial, ya que es necesario condicionar o colocar restricciones para las tareas, por esta razón, en esta sección se presentan las estructuras de control selectivas simple y compuesta.

1. DEFINICIÓN DE ESTRUCTURA DE CONTROL SELECTIVA SIMPLE

La estructura de control selectiva simple tiene un punto de decisión que evalúa una condición y si es verdadero, ejecuta un conjunto de acciones.[7]

Recuerde que, en caso de la condición al ser evaluada sea falsa, no debe realizar acción alguna, de lo contrario, dejaría de ser una estructura de control selectiva simple.

Si observa la figura 44, notará que la condición se expresa en un rombo, del cual salen dos flujos (flechas). Si es verdadero, el flujo continúa realizando un conjunto de acciones y, si es falso, el flujo no realiza acciones y ambos finalizan el recorrido de la estructura de control selectiva simple.

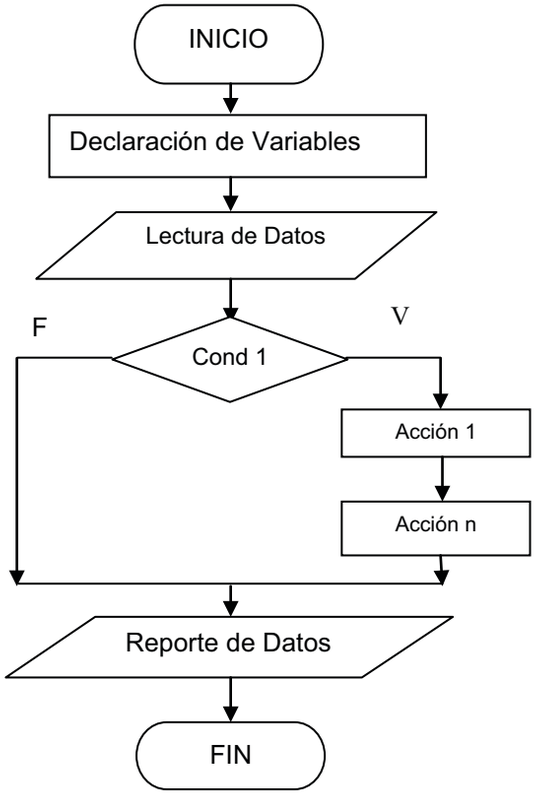


Figura N° 44: Diagrama de flujo para una Estructura de control selectiva simple
Fuente: Carol Rojas M.

7 Joyanes, L. (2008). Fundamentos de programación. Madrid: McGRAW-HILL

Ejemplo de Estructura selectiva simple:

Tenemos el requerimiento de sumar dos números enteros positivos:

$$c = a + b$$

Solución:

Se sugiere definir las variables para el proceso de entrada, asignándoles un nombre. En este caso, pueden ser cualquier número, por eso, se puede designar las variables:

Definición de variables:

a: valor del primer número entero positivo.

b: valor del segundo número entero positivo.

c: valor del resultado la suma de dos números **a** y **b**.

Solución en Diagrama de flujo:

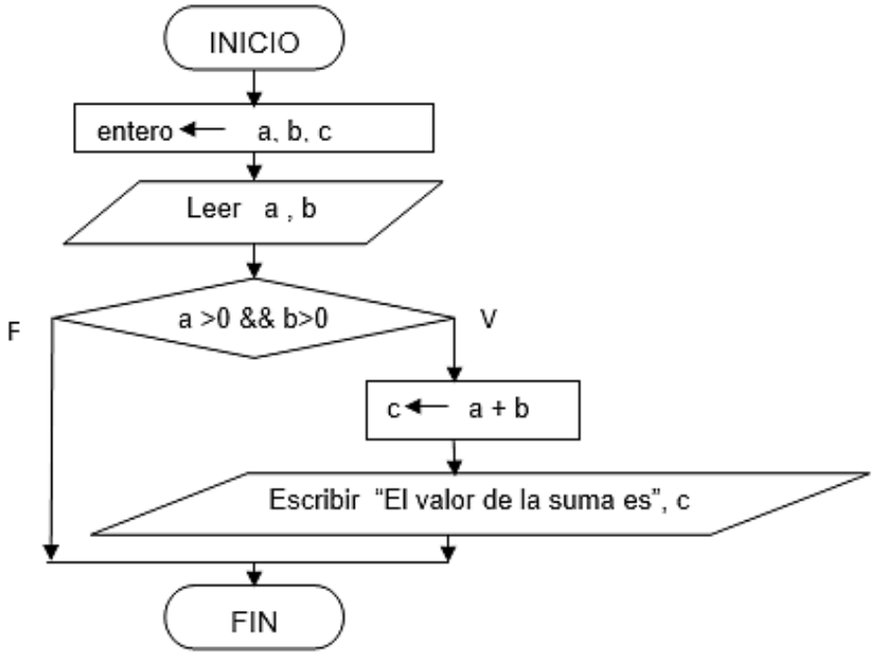


Figura N° 45: Solución de Diagrama de flujo para ejemplo de una Estructura de control selectiva simple
Fuente: Carol Rojas M.

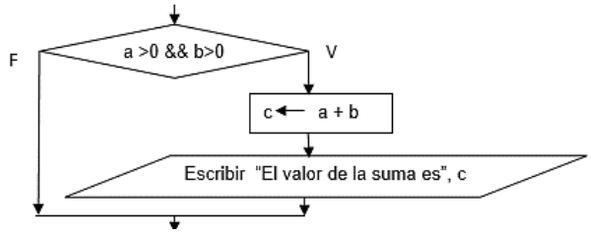
Observe, en el diagrama anterior, que la condición expresada en el rombo evalúa que ambas variables sean mayores que cero (usa el operador **&&** para la conjunción, en otros casos puede usar el operador **||** para la disyunción es decir o cumpla una u otra condición).

Solución en Código C/C++:

```
#include<iostream>
using namespace std;

int main( )
{
    float a, b, c;

    cout<<"Ingrese primer numero: ";
    cin>>a;
    cout<<"\n";

    cout<<"Ingrese segundo numero: ";
    cin>>b;
    cout<<"\n";

    if (a>0 && b>0)
    {
        c = a + b;
        cout<<"El valor la suma es: ";
        cout<<c;
        cout<<"\n";
    }

    return 0;
}
```

Figura N° 46: Solución en Código C/C++ para ejemplo de una Estructura de control selectiva simple
Fuente: Carol Rojas M.

Considerando el código C/C++ mostrado en la figura anterior, podemos comparar la representación en diagrama de flujo y su expresión en código.




```
if (a>0 && b>0)
{
    c = a + b;
    cout<<"El valor la suma es: ";
    cout<<c;
    cout<<"\n";
}
```

Figura N° 47: Diagrama de flujo y código C/C++ de una Estructura de control selectiva simple
Fuente: Carol Rojas M.

2. DEFINICIÓN DE ESTRUCTURA DE CONTROL SELECTIVA COMPUESTA

La estructura de control selectiva compuesta tiene un punto de decisión que evalúa una condición y si es verdadero, ejecuta un conjunto de acciones y si es falso ejecuta otro conjunto de acciones.

Si observa la figura 48, notará que la condición se expresa en un rombo, del cual salen dos flujos (flechas). Si es verdadero, el flujo continúa realizando un conjunto de acciones y, si es falso, también realiza otro grupo de acciones y ambos finalizan el recorrido de la estructura de control selectiva compuesta.

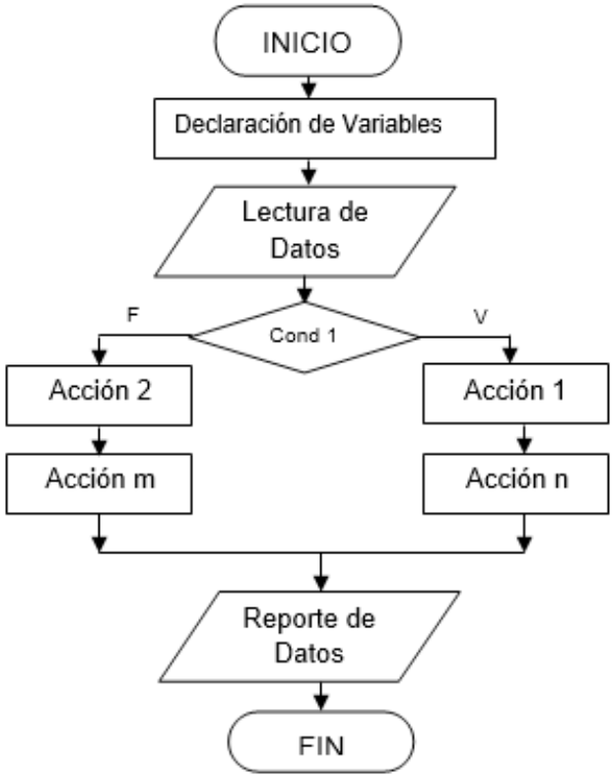


Figura N° 48:Diagrama de flujo para una Estructura de control selectiva compuesta
Fuente: Carol Rojas M.

Ejemplo de Estructura selectiva compuesta:

Tenemos el requerimiento de Sumar dos números enteros si son positivos, sino multiplicarlos.

Solución:

Se sugiere definir las variables para el proceso de entrada, asignándoles un nombre en este caso, pueden ser cualquier número, por eso se pueden designar las variables:

Definición de variables:

a: valor del primer número entero positivo.

b: valor del segundo número entero positivo.

c: valor del resultado la suma o multiplicación de dos números **a** y **b**.

Solución en Diagrama de flujo:

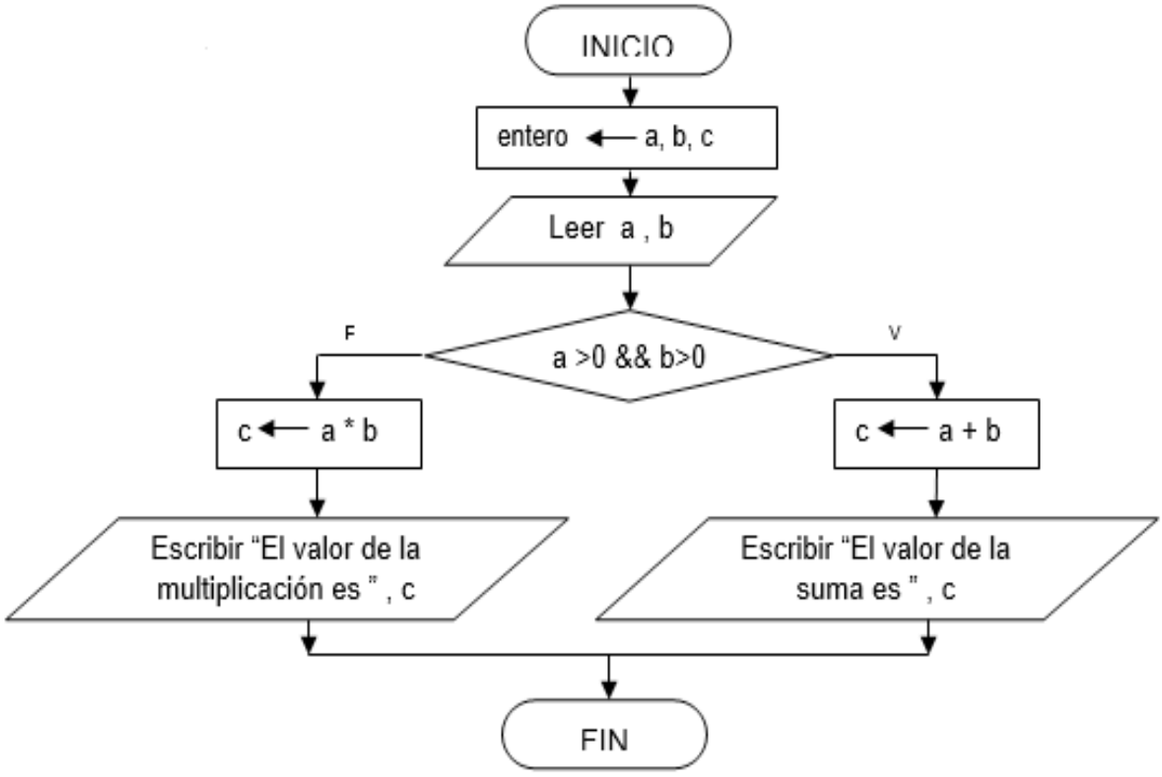


Figura N° 49: Solución de Diagrama de flujo para ejemplo de una Estructura de control selectiva compuesta
Fuente: Carol Rojas M.

Observe, en el diagrama anterior, que la condición expresada en el rombo evalúa que ambas variables sean mayores que cero para sumar.

Pero, si al menos una de las variables no cumple con la condición, se multiplicarán, y cada cálculo es asignado a la variable **c.**

Como el operador usado en la condición es de conjunción, nunca se realizarán los dos cálculos al mismo tiempo, por lo que ya no es necesario utilizar una variable distinta por cada cálculo de suma o de multiplicación.

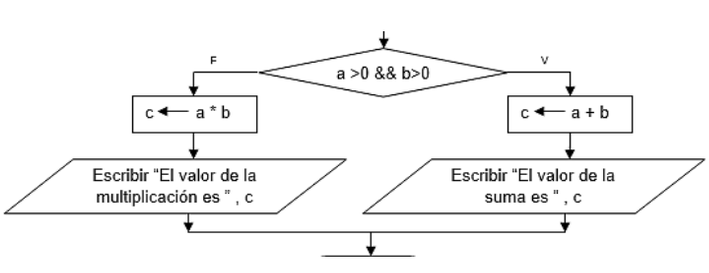
Es importante considerar el adecuado uso de las variables en el programa, para que, de esta manera, podamos alinearnos a una de las características de software que es la eficiencia en el uso de recursos de memoria del computador, además de la fácil corrección y el mantenimiento del programa.

Solución en Código C/C++:

```
#include<iostream>
using namespace std;

void main( )
{
    int a, b, c;

    cout<<"Ingrese primer número: ";
    cin>>a;
    cout<<"\n";

    cout<<"Ingrese segundo número: ";
    cin>>b;
    cout<<"\n";

    if(a>0 && b> 0)
    {
        c = a + b;
        cout<<"El valor la suma es: ";
        cout<<c;
        cout<<"\n";
    }
    else
    {
        c = a * b;
        cout<<"El valor la multiplicacion es: ";
        cout<<c;
        cout<<"\n";
    }
}
```

Figura N° 50: Solución en Código C/C++ para ejemplo de una Estructura de control selectiva compuesta
Fuente: Carol Rojas M.

Considerando el código C/C++ mostrado en la figura anterior, podemos comparar la representación en diagrama de flujo y su expresión en código.




```
if(a>0 && b> 0)
{
    c = a + b;
    cout<<"El valor la suma es: ";
    cout<<c;
    cout<<"\n";
}
else
{
    c = a * b;
    cout<<"El valor la multiplicacion es: ";
    cout<<c;
    cout<<"\n";
}
```

Figura N° 51: Diagrama de flujo y código C/C++ de una Estructura de control selectiva compuesta
Fuente: Carol Rojas M.

Ambas estructuras de control selectivas y compuestas pueden combinarse en la solución de un programa, dando como resultado el anidamiento de estas estructuras.

Ejemplo: Ingresar un número entero y, si es positivo y diferente de cero, reportar si se encuentra entre los diez primeros números.

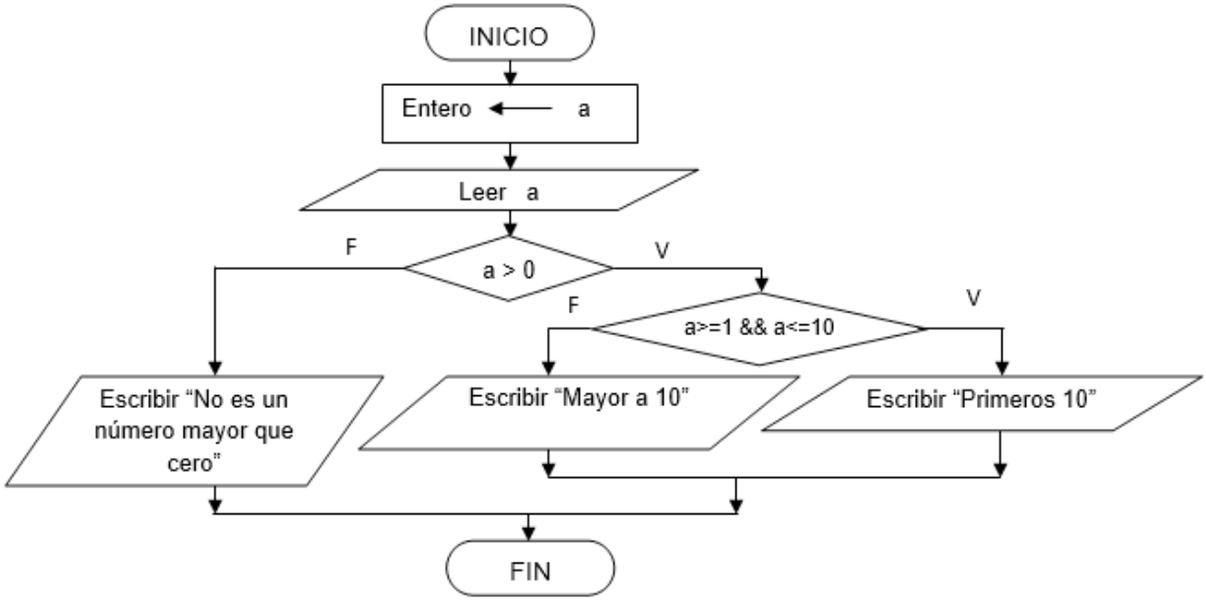


Figura N° 52: Diagrama de flujo para ejemplo de Estructura de control anidada
Fuente: Carol Rojas M.

Considerando el código C/C++ mostrado en la figura anterior, podemos comparar la representación en diagrama de flujo y su expresión en código.

```
1   #include<iostream>
2   using namespace std;
3
4   void main( )
5   {
6       int a;
7
8       cout<<"Ingrese número: ";
9       cin>>a;
10
11      if(a>0)
12          if(a>=1 && a<10)
13              cout<<"Primeros 10";
14          else
15              cout<<"Son más de los primeros 10";
16      else
17          cout<<"No es un número mayor que cero";
18  }
```

Figura N° 53: Solución en Código C/C++ para ejemplo de Estructura de control anidada
Fuente: Carol Rojas M.

LECTURA SELECCIONADA N° 1

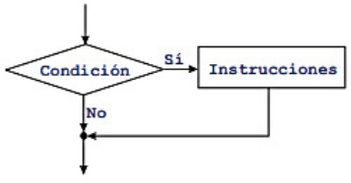
ESTRUCTURAS DE CONTROL: CONDICIONALES [8]

[8] Departamento de ecuaciones diferenciales y análisis numérico. Universidad de Sevilla. Ubicado en: http://departamento.us.es/edan/php/asig/LICFIS/LFIPC/Tema5FISPC0809.pdf

Estructura condicional simple: IF

Este es el tipo más sencillo de estructura condicional. Sirve para implementar acciones condicionales del tipo siguiente: Si se verifica una determinada condición, se debe ejecutar una serie de instrucciones y luego seguir adelante. Si la condición NO se cumple, NO se ejecutan dichas instrucciones y se sigue adelante.

```
...
if condición
   instrucciones
end
...
```



Obsérvese que, en ambos casos (que se verifique o no la condición), los "caminos" bifurcados se unen, posteriormente, en un punto, es decir, el flujo del programa recupera su carácter secuencial, y se continúa ejecutando por la instrucción siguiente a la estructura IF. Como ejemplo de utilización de este tipo de condicional, se considera el cálculo del valor en un punto x de una función definida por partes, como, por ejemplo:

$$\gamma(x) = \begin{cases} x_0 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

El algoritmo 5.3 muestra el seudocódigo correspondiente:

Algoritmo 5.3 Cálculo del valor de la función $f(x) = 0$ si $x \leq 0$, $f(x) = x^2$ si $x > 0$.

```
Inicio
   1- LEER x
   2- HACER f=0
   3- Si x>0
        HACER f=x^2
      Fin Si
   4- IMPRIMIR 'El valor de la funcion es: ', f
Fin
```

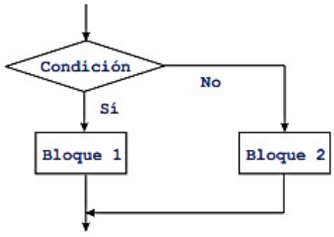
Estructura condicional doble: IF - ELSE

Este tipo de estructura permite implementar condicionales en los que hay dos acciones alternativas:

- Si se verifica una determinada condición, ejecutar una serie de instrucciones (bloque 1).
- Si no, esto es, si la condición NO se verifica, ejecutar otra serie de instrucciones (bloque 2).

En otras palabras, en este tipo de estructuras hay una alternativa: se hace una cosa o se hace la otra. En ambos casos, se sigue por la instrucción siguiente a la estructura IF - ELSE.

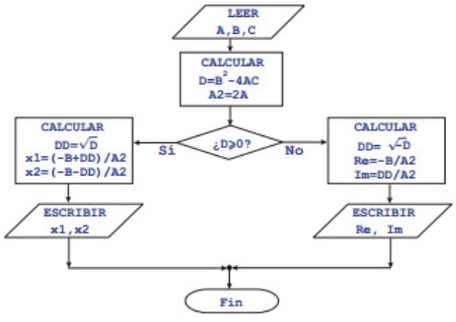
```
...
if condición
   bloque-1
else
   bloque-2
end
...
```



Como ejemplo de utilización de este tipo de estructuras, se plantea el problema de calcular las raíces de una ecuación de segundo grado.

$$ax_2 + px + c = 0$$

Distinguiendo dos casos: que las raíces sean reales o que sean complejas (no se contempla, de momento, distinguir entre una o dos raíces reales).



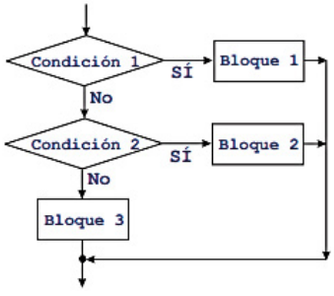
Estructura condicional múltiple: IF - ELSEIF - ELSE

En su forma más general, la estructura IF - ELSEIF - ELSE permite implementar condicionales más complicados, en los que se "encadenan" condiciones en la forma siguiente:

- Si se verifica la condición 1, ejecutar las instrucciones del bloque 1.
- Si no se verifica la condición 1, pero Sí se verifica la condición 2, ejecutar las instrucciones del bloque 2.
- Si no se ha verificado ninguna de las condiciones anteriores, ejecutar las instrucciones del bloque 3.

En cualquiera de los casos, el flujo del programa continúa por la instrucción siguiente a la estructura IF - ELSEIF - ELSE.

```
...
if condición-1
   bloque-1
elseif condición-2
   bloque-2
else
   bloque-3
end
...
```



Algoritmo 5.5 Determinación del signo de un número: positivo, negativo o nulo.

```
Inicio
   1- LEER X
   2- Si X>0
        IMPRIMIR 'El número tiene signo positivo'
      Si no, si X<0
        IMPRIMIR 'El numero tiene signo negativo'
      Si no
        IMPRIMIR 'El numero es nulo'
Fin
```

En la estructura IF - ELSEIF - ELSE se puede multiplicar la cláusula ELS EIF, obteniéndose así una "cascada" de condiciones, como se muestra en el organigrama, cuyo funcionamiento es claro. En este tipo de estructura condicional, la cláusula ELSE junto con su bloque de instrucciones puede no estar presente. Las distintas estructuras condicionales descritas pueden ser anidadas, es decir, puede incluirse una estructura IF (de cualquier tipo), como parte de las instrucciones que forman el bloque de uno de los casos de otro IF. Como es lógico, no puede haber solapamiento. Cada estructura IF debe tener su propio fin (end).

SINTESIS del TEMA I- II

Estructura de Control Selectiva

Permite decidir el conjunto de acciones a realizar:

Puede ser Selectiva Simple:

```
if(condición)
{

}
```

Puede ser Selectiva Compuesta:

```
if(condición)
{

}
else
{

}
```

TEMA N° 3:
ESTRUCTURA DE CONTROL SELECTIVA: MÚLTIPLE

Estimado estudiante, hasta este punto se ha evaluado condiciones que permiten resultados en verdadero o falso, pero no necesariamente siempre se tendrá estos casos.

A continuación, se presenta una estructura de control de programación, que le permite evaluar un caso, de un conjunto de opciones.

1. DEFINICIÓN DE ESTRUCTURA DE CONTROL SELECTIVA MÚLTIPLE

La estructura de control selectiva múltiple tiene un punto de decisión que evalúa una condición y si es verdadero, ejecuta un conjunto de acciones, según el caso.

Si observa la figura 54, notará que la condición se expresa en un rombo, del cual salen tantos flujos (flechas) como casos se tenga. Si uno de los casos coincide, según la condición dada, se ejecuta un conjunto de acciones y si no coincide el caso, se puede enviar un mensaje por defecto.

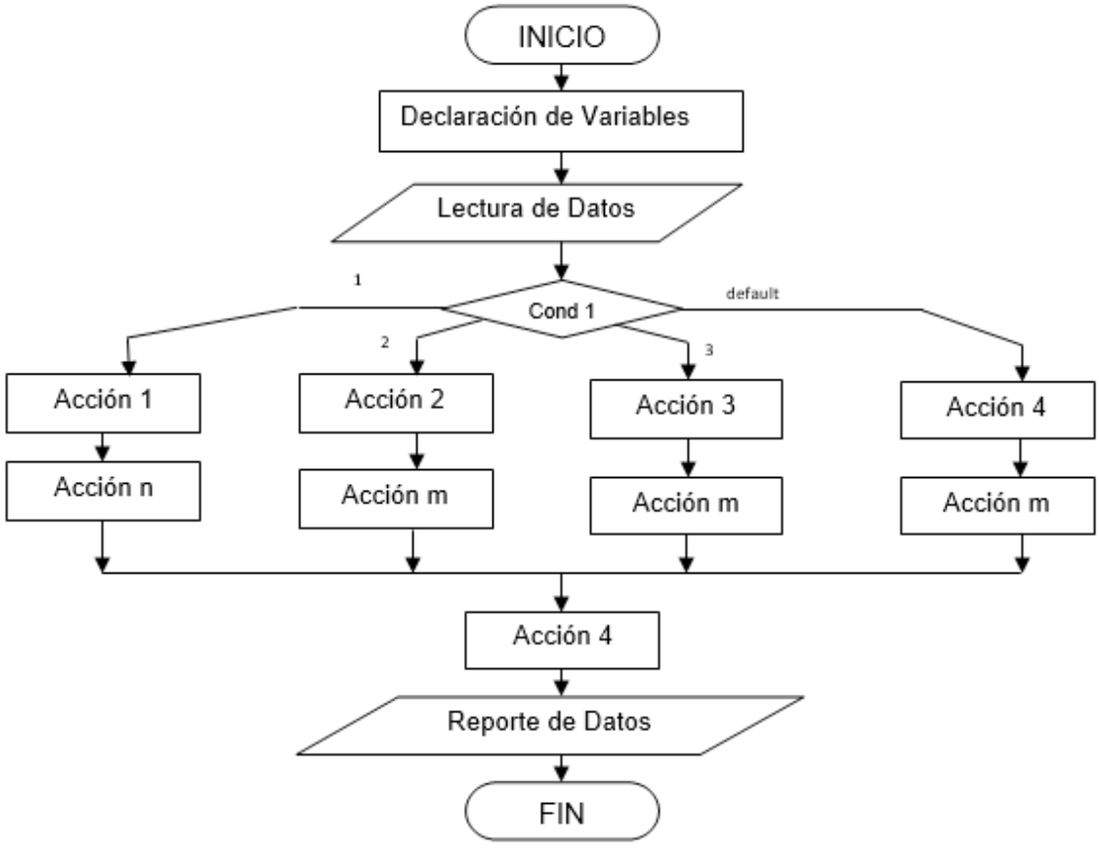


Figura N° 54: Diagrama de flujo para una Estructura de control selectiva múltiple
Fuente: Carol Rojas M.

2. EJEMPLO DE ESTRUCTURA SELECTIVA MÚLTIPLE

Tenemos el requerimiento de ingresar un número del uno al cinco y reporte la vocal que represente.

Solución:

Se sugiere definir las variables para el proceso de entrada, asignándoles un nombre que represente al número ingresado:

DEFINICIÓN DE VARIABLES:

num: valor del numero entero positivo.

Solución en Diagrama de flujo:

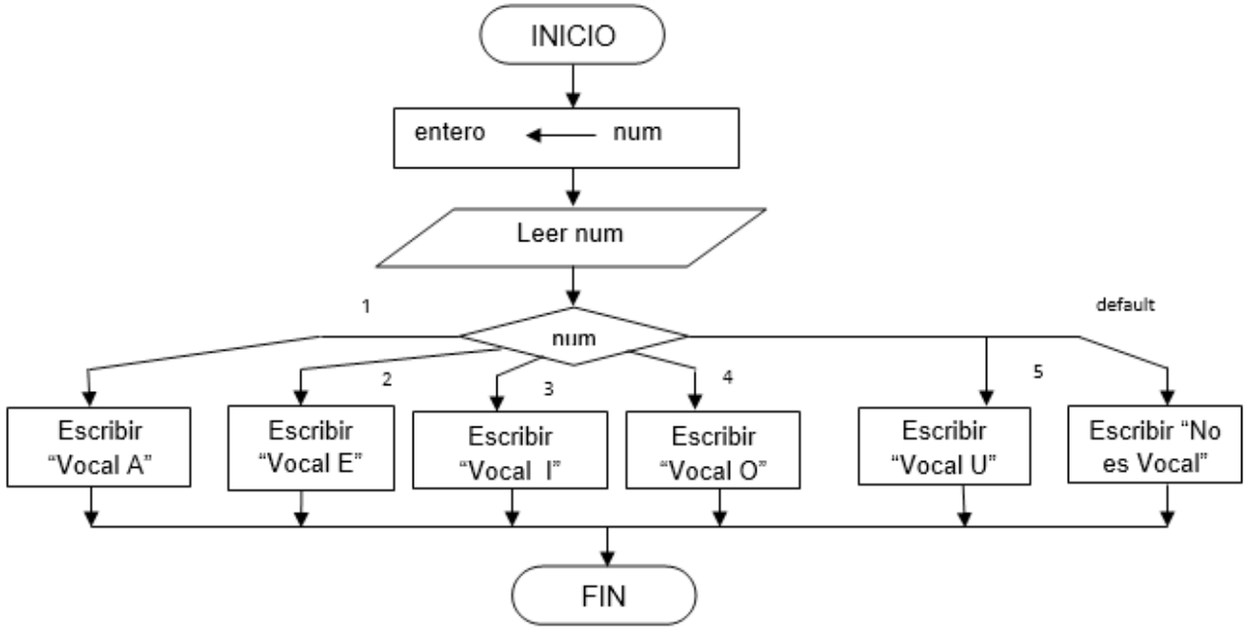


Figura N° 55: Solución de Diagrama de flujo para ejemplo1 de una Estructura de control selectiva compuesta
Fuente: Carol Rojas M.

Observe en el diagrama anterior que, la condición expresada en el rombo permite el ingreso de un solo valor (no permite un rango de valores por ejemplo >0) y, según la coincidencia del caso con el valor ingresado, se ejecuta el conjunto de acciones, finalizando la estructura de control múltiple, y no permitiendo la ejecución de otro caso (usando break).

Solución en Código C/C++:

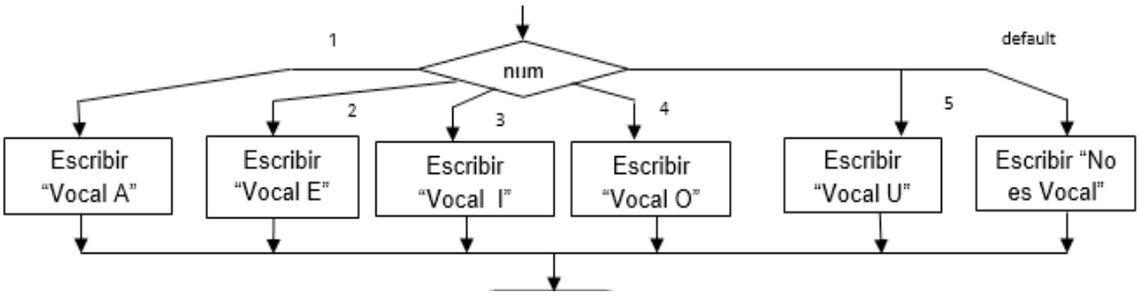
```
#include<iostream>
using namespace std;

void main( )
{
    int num;

    cout<<"Ingrese número: ";
    cin>>num;

    switch(num)
    {
        case 1: {
                  cout<<"Vocal A";
                }break;

        case 2: {
                  cout<<"Vocal E";
                }break;

        case 3: {
                  cout<<"Vocal I";
                }break;

        case 4: {
                  cout<<"Vocal O";
                }break;

        case 5: {
                  cout<<"Vocal U";
                }break;

        default: cout<<"No es numero para una vocal";
    }
}
```

Figura N° 56: Solución en Código C/C++ para ejemplo1 de una Estructura de control selectiva múltiple
Fuente: Carol Rojas M.

Si observa la figura, la variable "num", es de tipo entero, es decir (ejemplo: int num;), por lo que los casos evalúan ese valor como entero (ejemplo: case 1:) sin usar comilla simple.

Si se hubiese definido como carácter (ejemplo: char num;), la evaluación del caso hubiese sido considerando la comilla simple (ejemplo: case '1').

Considerando el código C/C++ mostrado en la figura anterior, podemos comparar la representación en diagrama de flujo y su expresión en código.




```
switch(num)
{
    case 1: {
              cout<<"Vocal A";
            }break;

    case 2: {
              cout<<"Vocal E";
            }break;

    case 3: {
              cout<<"Vocal I";
            }break;

    case 4: {
              cout<<"Vocal O";
            }break;

    case 5: {
              cout<<"Vocal U";
            }break;

    default: cout<<"No es numero para una vocal";
}
```

Figura N° 57: Diagrama de flujo y código C/C++ de una Estructura de control selectiva compuesta
Fuente: Carol Rojas M.

Mostramos otro ejemplo, de Estructura de control selectiva múltiple:

Se solicita ingresar la inicial (mayúscula o minúscula) de un tipo de barco y reporte el nombre del mismo, y si no existe, reportar "TIPO DE BARCO NO ENCONTRADO".

Recuerde que el switch solo evalúa un valor dado y ejecuta directamente el caso en seguir buscando en los demás casos gracias a la instrucción break, que interrumpe el flujo del programa, en este caso del switch.

Pero debe preguntarse, ¿qué pasaría si obvia la instrucción break de algún caso? Lógicamente no se rompe el flujo del switch, por lo que procesaría por defecto el siguiente caso con sus respectivas instrucciones.

Esto constituye una oportunidad para dar solución a este nuevo ejemplo, sin necesidad de emplear funciones especiales para diferenciar mayúsculas y minúsculas.

Solución:

Definimos variables, con un cuadro que le permita definir las tres partes del algoritmo de solución:

Cuadro N° 6: Solución con partes del algoritmo para el ejemplo 3 de una Estructura de control secuencial

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

tbarco	tbarco	switch(tbarco)
		{
		case 'b':
		case 'B': cout<<"El tipo es Buque"; break;
		case 'c':
		case 'C': cout<<"El tipo es Crucero"; break;
		case 'f':
		case 'F': cout<<"El tipo es Fragata"; break;
		default: cout<<"No es un tipo de barco";
		}
			tbarco

Fuente: Carol Rojas M.

Solución en Código C/C++:

```
#include<iostream>
using namespace std;

int main()
{
    char tbarco;

    cout<<"Ingrese tipo de barco: ";
    cin>>tbarco;

    switch(tbarco)
    {
        case 'b':
        case 'B': cout<<"El tipo es Buque"; break;

        case 'c':
        case 'C': cout<<"El tipo es Crucero"; break;

        case 'f':
        case 'F': cout<<"El tipo es Fragata"; break;

        default: cout<<"No es un tipo de barco";
    }

return 0;
}
```

Figura N° 58: Solución en Código C/C++ para el ejemplo 2 de una Estructura de control selectiva múltiple
Fuente: Carol Rojas M.

Un tercer ejemplo, combina el uso de las estructuras de control selectivas, tanto simple, compuesta y múltiple.

Se requiere calcular el monto total de pago, luego de ingresar el nombre y éenero de un cliente, además de la cantidad de productos a adquirir, con su respectivo precio.

Para calcular el monto total a pagar, se asignan descuentos según el género y la cantidad de productos adquiridos, como se muestra en la siguiente tabla:

Cuadro N° 7: Tabla de descuentos para el ejemplo

	GÉNERO			
	M		F	
Cantidad	<=10	>10	<=10	>10
Porcentaje de descuento	0.2	0.5	0.3	0.4

Fuente: Carol Rojas M.

Solución:

Definimos variables con un cuadro que le permita definir las tres partes del algoritmo de solución.

Observe que se definen todas las variables a usar tanto en el ingreso, proceso y salida.

Además, observa que en el proceso, se solicita ingresar la cantidad de producto, validando su ingreso con una estructura de control selectiva compuesta, garantizando que sea positivo.

También se solicita ingresar el género del cliente y, de igual forma, se valida su ingreso con una estructura de control selectiva compuesta, garantizando que solo se ingrese las iniciales de masculino y femenino en minúscula.

Luego, se usa una estructura de control selectiva múltiple para validar por casos el género del cliente y realizar la asignación del porcentaje de descuento, de acuerdo con la cantidad de productos, verificada en una estructura de control selectiva compuesta.

Se procede a calcular el monto de pago, y se recomienda hacerlo por partes, a fin de tener mayor control de los datos en las variables.

Finalmente, se procede a reportar o mostrar en pantalla el valor calculado del pago, según el requerimiento del usuario.

Estimado estudiante, le recomiendo seguir este procedimiento de solución en los demás casos propuestos, es decir, validando el ingreso y calculando por etapas, los valores finales de la variable.

Cuadro N° 8: Solución con partes del algoritmo para el ejemplo 3 de una Estructura de control selectiva combinadas

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

nombClie
geneClie
precioProd
cantProd
PorcDscto
MontoBruto
MontoDscto
MontoPago
	nombClie
	cantProd
	geneClie
		Si(cantProd>0)
		{ Leer geneClie;
		si(geneClie == 'm' || geneClie == 'M' ||
		geneClie == 'f' || geneClie == 'F')
		{ Si(geneClie)
		{ caso 'm':
		caso 'M':
		{
		Si(cantProd <= 10)
		PorcDscto = 0.2;
		Sino
		if(cantProd > 10)
		PorcDscto = 0.5;
		}break;
		caso 'f':
		caso 'F':
		{
		Si(cantProd <= 10)
		PorcDscto = 0.3;
		Sino
		if(cantProd > 10)
		PorcDscto = 0.4;
		}break;
		}
		MontoBruto = cantProd * precioProd;
		MontoDscto = MontoBruto * PorcDscto;
		MontoPago = MontoBruto - MontoDscto;
		Escribir "El monto bruto es: "
		Escribir MontoBruto
		Escribir "El monto descuento es: "
		Escribir MontoDscto
		Escribir "El monto Pago es: "
		Escribir MontoPago
		}
		Sino
		Escribir "ERROR. Debe ingresar F o M."
		}
		Sino
		cout<<"ERROR. Debe ingresar mayor a cero.";
			MontoBruto
			MontoDscto
			MontoPago

Fuente: Carol Rojas M.

```cpp
#include<iostream>
#include<string>
using namespace std;

int main()
{
    string nombClie;
    char geneClie;
    double precioProd=100;
    int cantProd;
    double PorcDscto, MontoBruto, MontoDscto, MontoPago;

    cout<<"Ingrese Nombre de Cliente: ";
    cin>>nombClie;

    cout<<"Ingrese Cantidad de Productos: ";
    cin>>cantProd;

    //Validar cantidad mayor a cero
    if(cantProd>0)
    {
        cout<<"Ingrese Genero de Cliente: ";
        cin>>geneClie;

        //Validar que se ingrese genero
        if(geneClie == 'm' || geneClie == 'M' || geneClie == 'f' || geneClie == 'F')
        {
            switch(geneClie)
            {
                case 'm':
                case 'M':
                        {
                            if(cantProd <= 10)
                                PorcDscto = 0.2;
                            else
                                if(cantProd > 10)
                                    PorcDscto = 0.5;
                        }break;

                case 'f':
                case 'F':
                        {
                            if(cantProd <= 10)
                                PorcDscto = 0.3;
                            else
                                if(cantProd > 10)
                                    PorcDscto = 0.4;
                        }break;

            }

            MontoBruto = cantProd * precioProd;
            MontoDscto = MontoBruto * PorcDscto;
            MontoPago = MontoBruto - MontoDscto;

            cout<<"El monto bruto es: ";
            cout<<MontoBruto<<endl;
            cout<<"El monto descuento es: ";
            cout<<MontoDscto<<endl;
            cout<<"El monto Pago es: ";
            cout<<MontoPago<<endl;
        }
        else
            cout<<"ERROR. Debe ingresar femenino o masculino.";
    }
    else
        cout<<"ERROR. Debe ingresar mayor a cero.";

    return 0;
}
```

Figura N° 59: Solución en Código C/C++ para el ejemplo 3 de una Estructura de control selectiva combinadas
Fuente: Carol Rojas M.

SINTESIS del TEMA III

Estructura de Control Selectiva

También puede ser Múltiple:

Permite decidir entre varios casos:

```
switch(valor)
 {
    case 1:
                {

                } break;
    case 2:
                {

                } break;
   case 3:
                {

                } break;

  }
```

Puede combinarse con Selectiva simple y Compuesta:

ACTIVIDAD FORMATIVA Nº 2

Elabora el programa de cómputo para cada situación propuesta.

INSTRUCCIONES:

1. Lee y analiza los temas N° 2 y N° 3 y extrae las ideas fundamentales de la estructura de control secuencial para la programación.
2. Observe el vídeo **"La estructura condicional simple/compuesta (if) / C++"** y complemente la información obtenida de los temas N° 2 y N° 3.
3. Elabore un programa en lenguaje C/C++ para las siguientes situación problema:

3.1 En una empresa, de acuerdo con el género del trabajador, se asigna una bonificación.

Reportar el monto de la bonificación y el monto total de pago de un trabajador.

** Si es F: es 20% sobre el monto de pago.

** Si es M: es 18% sobre el monto de pago.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

3.2 En un banco, a los clientes que son de tipo 1, 2, 3, y que aperturan cuentas de ahorro(A), corriente (C), o valores (V), todas en soles, se les otorga un bono, endólares, de acuerdo con la siguiente tabla:

CUENTA: TIPO CLIENTE	A	C	V
1	$ 500	$ 400	$ 300
2	$ 400	$ 300	$ 200
3	$ 300	$ 200	$ 100

Reportar el tipo de cliente, la cuenta que apertura, el monto total de la cuenta. Considere el ingreso del tipo de cambio de moneda al día.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

El instrumento para calificar es una Lista de cotejo con los siguientes criterios, para un archivo en Word con la tabla de las partes del algoritmo, y el archivo fuente (.cpp) de cada ejercicio, en una carpeta ApellidoNombreAlumno.zip:

CRITERIO	PUNTAJE	
	EJERCICIO 3.1	EJERCICIO 3.2
Elaboración de la tabla con las tres partes del algoritmo y las variables a usar.	1	1
Declarar las variables con su respectivo tipo de dato.	1	1
Validar cada dato ingresado con la sentencia selectiva compuesta.	1	3
Usar la sentencia selectiva múltiple para el desarrollo de cada caso.	2	3
Realizar los cálculos, según cada condición.	1	2
Mostrar los datos solicitados, según la tabla de las partes del algoritmo.	2	2

GLOSARIO DE LA UNIDAD II

A

ACCIÓN
Sentencia de programación, puede ser secuencial, selectiva o repetitiva.

B

BLOQUE
En programación, un bloque es un conjunto de instrucciones o sentencias que pueden ser las estructuras de control secuencial, selectiva o repetitiva.

C

CARACTER
Tipo de dato de un lenguaje de programación, que solo permite un símbolo o una letra.

CLÁUSULA
En programación, es una palabara reservada, es decir, una instrucción con una sintáxis específica, según el lenguaje de programación.

R

REQUERIMIENTO
Solicitud o necesidad que tiene un cliente o usuario del programa, para ser solucionado con un programa de cómputo.

RESTRICCIÓN
Expresión condicional que refleja lo especificado por el cliente, como requerimiento del proceso a desarrollar.

S

SOLAPAMIENTO
Tiempo de uso de dos o más estructuras de control de programación, al mismo tiempo.

SWTICH
Es la forma de escribir, o sintáxis, de la estructura selectiva múltiple, y se puede entender como una compuerta que solo permite ingresar un valor y no un rango de valores.

BIBLIOGRAFÍA DE LA UNIDAD II

Carrasco, A. (2005). *Principios de programación. Algoritmos y su creación en C++.* Perú: AC Editores.

Joyanes, L. (2008). *Fundamentos de programación.* Madrid: McGRAW-HILL.

AUTOEVALUACIÓN DE LA UNIDAD II

1. Las estructuras de control o sentencias básicas, como técnicas de la programación estructurada son:

   A) Algoritmos, selectivas, repetitivas

   B) Diagramas, secuenciales, repetitivas

   C) Secuenciales, algoritmos, repetitivas

   D) Diagramas, algoritmos, repetitivas

   E) Secuenciales, selectivas, repetitivas

2. Indique la alternativa que describa el uso de una estructura de control secuencial.

   A) Es una sentencia que tiene dos decisiones.

   B) Es una sentencia que no repite ni condiciona.

   C) Es una sentencia repetitiva del algoritmo.

   D) Es una sentencia que tiene decisiones múltiples.

   E) Es una sentencia que tiene una decisión.

3. Indique la alternativa que describa el uso de una estructura de control selectiva simple.

   A) Es una sentencia repetitiva del algoritmo.

   B) Es una sentencia que tiene decisiones múltiples.

   C) Es una sentencia que tiene dos decisiones.

   D) Es una sentencia que tiene una decisión.

   E) Es una sentencia que no repite ni condiciona.

4. Indique la alternativa que describa el uso de una estructura de control selectiva compuesta.

   A) Es una sentencia que tiene dos decisiones.

   B) Es una sentencia repetitiva del algoritmo.

   C) Es una sentencia que tiene decisiones múltiples.

   D) Es una sentencia que tiene una decisión.

   E) Es una sentencia que no repite ni condiciona.

5. Indique la alternativa que describa el uso de una sentencia estructura de control múltiple.

A) Es una sentencia que tiene una decisión.

B) Es una sentencia que tiene dos decisiones.

C) Es una sentencia que no repite ni condiciona.

D) Es una sentencia repetitiva del algoritmo.

E) Es una sentencia que tiene varias decisiones.

6. Dado el siguiente bloque de código, indique el tipo de sentencia:

```
if(num>0)
      if(num >30 && num < 60 )
            cout<<pow(num,2);
```

A) Sentencia repetitiva para.

B) Sentencia selectiva compuesta.

C) Sentencia selectiva anidada.

D) Sentencia repetitiva hacer - mientras.

E) Sentencia selectiva múltiple.

7. Indique lo que realiza la instrucción "break".

A) Permite continuar con el flujo del programa.

B) Interrumpe la diagramación del algoritmo.

C) Permite con la diagramación del algoritmo.

D) Interrumpe las instrucciones secuenciales.

E) Interrumpe el flujo normal del programa.

8. Indique qué estructura de control aplica la instrucción "break".

A) Múltiple

B) Simple

C) Anidada

D) Compuesta

E) Secuencial

9. Para evaluar un valor de tipo de dato char, en una condición se usa:

   A) Comilla doble

   B) Llaves

   C) Corchetes

   D) Barra inclinada

   E) Comilla simple

10. Indique en qué estructura de control, no se puede condicionar rango de valores como, por ejemplo: >=10

   A) Simple

   B) Compuesta

   C) Secuencial

   D) Múltiple

   E) Anidada

UNIDAD III

"ESTRUCTURAS DE CONTROL PARA LA PROGRAMACIÓN: REPETITIVAS"

DIAGRAMA DE PRESENTACIÓN DE LA UNIDAD III

- Al finalizar la unidad, el estudiante será capaz de elaborar programas computacionales, aplicando las estructuras de control repetitivas.

CONTENIDOS	ACTIVIDADES FORMATIVAS (HABILIDADES Y ACTITUDES)	SISTEMA DE EVALUACIÓN (TÉCNICAS Y CRITERIOS)
Tema Nº 1: Estructura de control repetitiva mientras: 1 Definición de la Estructura de control repetitiva mientras **Tema Nº2: Estructura de comtrol repetitiva hacer-mientras:** 1 Definición de la Estructura de control repetitiva hacer-mientras **Tema Nº3: Estructura de control repetitiva des-de/por:** 1 Definición de la Estructura de control repetitiva desde/por	• Lee y analiza el tema Estructuras de control para la programación, repetitiva mien-tras. Observa el vídeo **Tutorial C++ 13. While y Do While** y elabora un pro-grama en lenguaje C/C++ por cada situación propuesta. • Lee y analiza el tema Estructuras de control para la programación, repetitiva hacermientras. Observa el vídeo **Tutorial programación C++ -Clase 7 bucles - do while** y elabora un programa en lenguaje C/C++ por cada situación propuesta complementando con los conceptos según la lectura N° 1. • Lee y analiza el tema Estructuras de control para la programación, repetitiva desde/por. Observa el vídeo **Tutorial programación C++ -Clase 7 bucles – for** y elabora un programa en lenguaje C/C++ por cada situación propuesta.	**Procedimientos e indicadores de evaluación permanente** • Entrega puntual de trabajos realizados. • Calidad, coherencia y pertinencia de contenidos desarrollados. • Prueba teórico-práctica, individual. • Actividades desarrolladas en sesiones tutorizadas. **Criterios de evaluación para diagrama de representación:** • Cuadro de identificación de partes del algoritmo: entrada, proceso, salida. • Programas elaborados con estructuras repetitivas en un lenguaje de programación propuesto, considerando las partes del algoritmo.

RECURSOS:

VIDEOS:

Tema Nº 1: (desde minuto 0:37 hasta 2:48 min)

Tutorial C++ 13. While y Do While

Ubicado en: Ubicado en: https://www.youtube.com/watch?v=MRiBUpgn-

Tema Nº 2: (hasta 2:56 min)

Tutorial programación C++ -Clase 7 bucles - do while

Ubicado en: https://www.youtube.com/watch?v=znSgaCnGGDU

Tema Nº 3: (hasta 3:03 min)

Tutorial programación C++ -Clase 7 bucles - for

Ubicado en: https://www.youtube.com/watch?v=8pGZj0PiOV8

DIAPOSITIVAS ELABORADAS POR EL DOCENTE:

Lectura complementaria:

Lectura seleccionada N° 1

Alcaraz, R. Estructuras de repetición. Una visión práctica de la Programación en C. México. Ubicada en:

https://ruidera.uclm.es/xmlui/bitstream/handle/10578/43/programacion_en_C.pdf?sequence=1

INSTRUMENTO DE EVALUACIÓN	• Lista de cotejo N° 2
BIBLIOGRAFÍA (BÁSICA Y COMPLEMENTARIA)	**Básica** **Joyanes, L. (2008).** Fundamentos de programación. 4a Edición. España: McGRAW-HILL. **Complementaria** **Carrasco, L. (2005).** Principios de programación. Algoritmos y su creación en C++. Perú: AC Editores. **Marcelo, R. (2014).** Fundamentos de programación C++. Lima-Perú: Editorial Macro.
RECURSOS EDUCATIVOS DIGITALES	**Alcaraz,R. (2009).** Una visión práctica de la programación en C. Recuperado el diciembre de 2015, de https://ruidera.uclm.es/xmlui/bitstream/handle/10578/43/programacion_en_C.pdf?sequence=1 **Moreira, R** Estructura Repetitiva y Ejemplos (20 de junio de 2015). Obtenido de http://programoreira.blogspot.pe/2014/05/estructura-repetitiva-y-ejemplos.html ,

TEMA N° 1:
ESTRUCTURA DE CONTROL REPETITIVA MIENTRAS

Estimado estudiante, habrá observado, que los programas requieren dar mejor facilidad de uso, por lo que en esta sección se presentan aquellas estructuras de control que le permitan condicionar y repetir el algoritmo de solución, en caso lo requiera el proceso o el usuario.

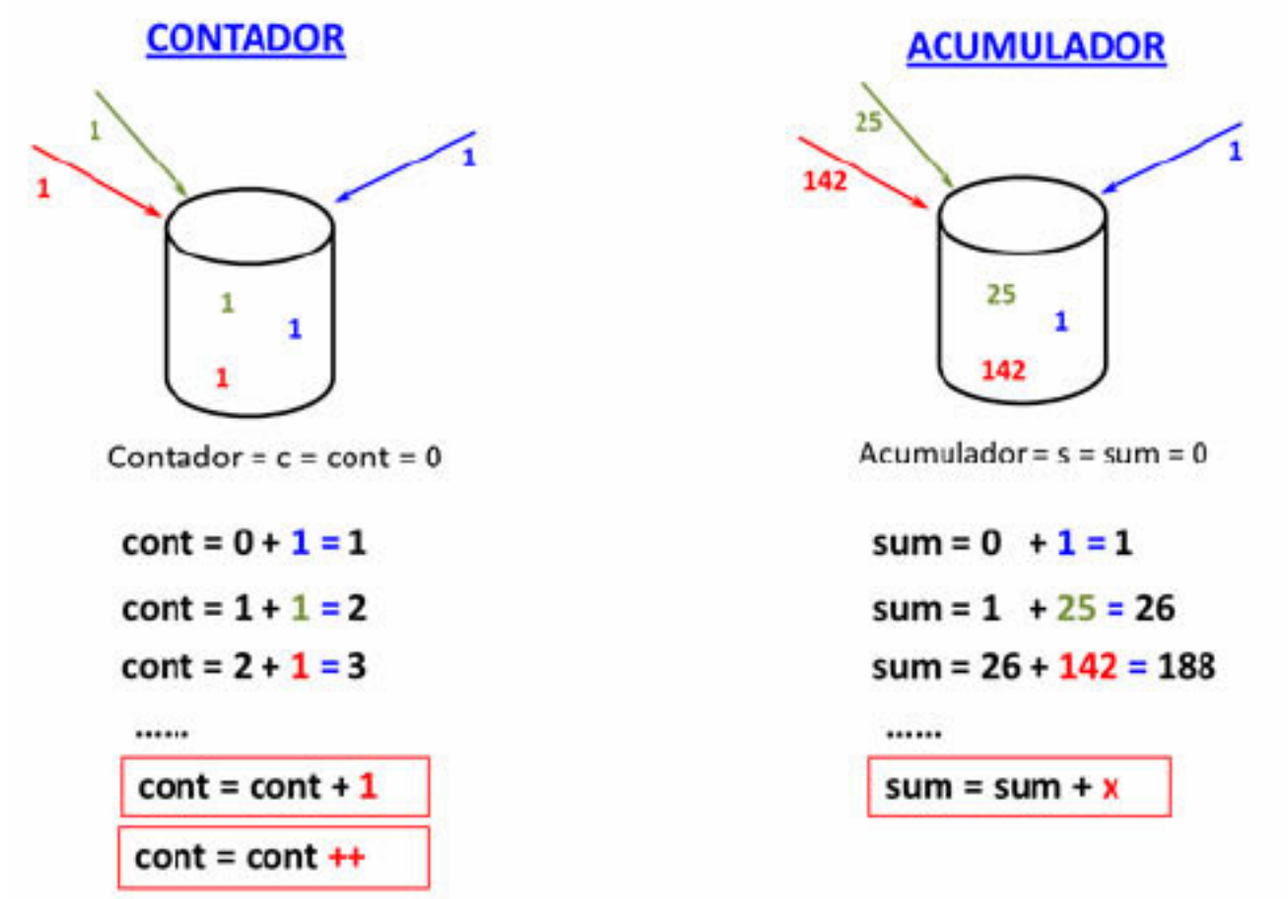
Para esta sección, es necesario tener en cuenta dos elementos de programación, que ayudarán a la aplicación de las estructuras repetitivas, nos referimos al contador y al acumulador.

El contador y el acumulador puede considerarlos como variables o espacio de memoria inicializados en cero (imagínelos como repositorios vacíos), que van modificando su valor según los datos que se le asignen.

La diferencia entre ambos, es que el contador está siendo modificado con las misma razón de valor (por ejemplo, se incrementa de uno en uno), mientras que el acumulador no necesariamente tiene la misma razón de valor de cambio, es decir el incremento se puede dar con un valor "x" cualquiera.

En la siguiente figura, se muestra una representación gráfica de ambos, con los respectivos valores asignados.

Figura 60
Ejemplo de contador y acumulador

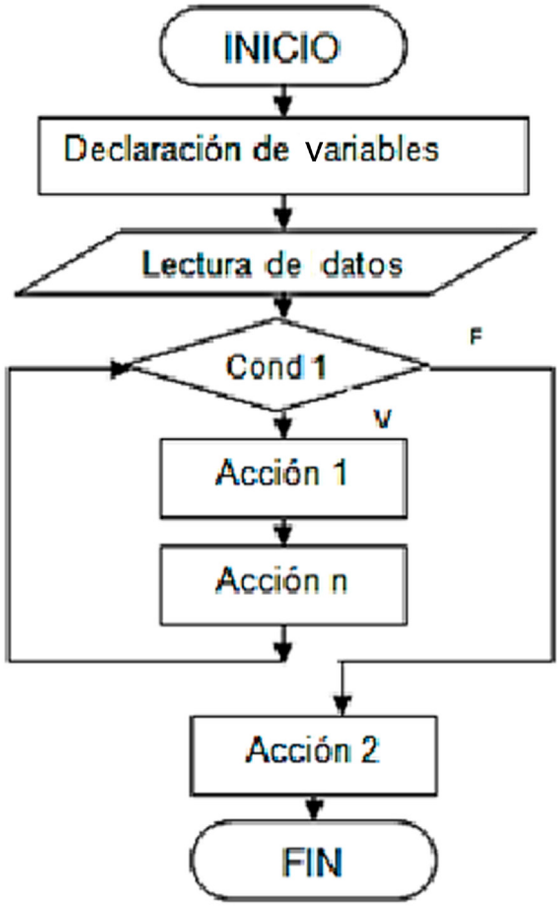


Fuente: Carol Rojas M.

1. DEFINICIÓN DE LA ESTRUCTURA DE CONTROL REPETITIVA MIENTRAS

Es una estructura que evalúa una expresión condicional y, si es verdadera, permite realizar el conjunto de acciones y regresa evaluar nuevamente la expresión condicional, hasta que deje de cumplir (Falso). [9]

Figura 61
Diagrama de flujo para una Estructura de control repetitiva mientras



Fuente: Carol Rojas M.

Ejemplo de Estructura repetitiva mientras:

Tenemos el requerimiento de imprimir la serie Fibonacci, menor a un límite dado:

Solución:

Se sugiere definir las variables, según la serie matemática:

Definición de variables:

a : valor inicial de la serie, inicializada con cero

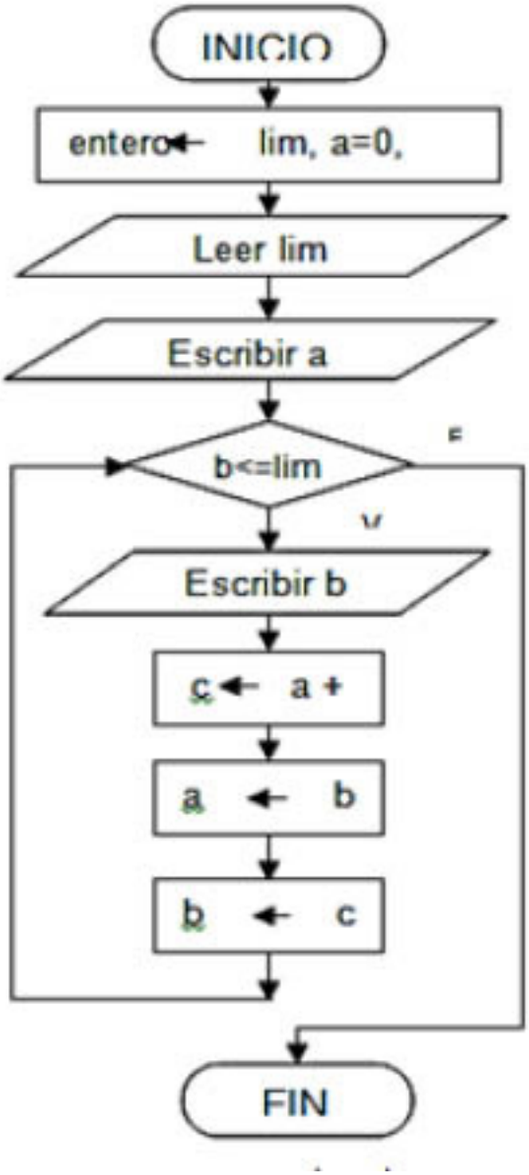
9 Joyanes, L. (2008). Fundamentos de programación. Madrid: McGRAW-HILL.

b : valor inicial de la serie, inicializada con uno

lim : valor máximo definido para el conjunto de números de la serie

Solución en Diagrama de flujo:

Figura 62
Solución de Diagrama de flujo para Ejemplo1 de una Estructura de control repetitiva mientras



Fuente: Carol Rojas M.

Solución en Código C/C++:

Figura 63
Solución en Código C/C++
para Ejemplo1 de una Estructura de control repetitiva mientras

```
#include<iostream>
using namespace std;

void main( )
{
    int lim, a=0, b=1,c;

    cout<<"Ingrese límite: ";
    cin>>lim;

    cout<<a;

    while(b<=lim)
    {
        cout<<b;
        c=a+b;
        a=b;
        b=c;
    }
}
```

Fuente: Carol Rojas M.

Un ejemplo de Estructura repetitiva mientras:[10]

Para ejemplificar el uso del contador y acumulador, solicitaremos hallar la suma y el promedio de los diez primeros números naturales positivos.

Definimos variables, con un cuadro que le permita definir las tres partes del algoritmo de solución.

Cuadro 9
Solución con partes del algoritmo
para Ejemplo2 de una Estructura de control repetitiva mientras

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

i
sumNumeros
promNumeros
		i = 1;
		Mientras (i <= 10)
		{
		sumNumeros = sumNumeros + i
		i = i+1
		}
		
		promNumeros = sumNumeros / 10;
		Escribir "La sumatoria de números es: "
		Escribir sumNumeros
		Escribir "El promedio de los números es: "
		Escribir promNumeros
			promCalif
			contAprob
			contDsprob

Fuente: Carol Rojas M.

Observe que en la solución propuesta, no se requiere ingresar un valor, por que se refiere a los diez primeros números naturales, que lógicamente están definidos.

10 Marcelo, R. (2014). Fundamentos de programación C++. Perú: Editorial Macro.

Pero si es necesario acumular o sumar cada número recorrido, que se incrementa con la estructura de control Mientras.

El incremento o recorrido de cada número natural positivo se realiza gracias al contador, definida en este ejemplo con la variable "i", que inicializa su valor en cero, y va contando de uno en uno, el mismo que es acumulado.

El valor acumulado, es que permite calcular el promedio de los números, después de haber finalizado la Estructura de control repetitiva mientras.

Solución en Código C/C++:

Figura 64
Solución en Código C/C++ para Ejemplo2 de una Estructura de control repetitiva mientras

```
#include<iostream>
using namespace std;

int main()
{
    int i;
    int sumNumeros = 0;
    double promNumeros;

    i = 1;  // contador de numeros

    while (i <= 10)
    {
        sumNumeros = sumNumeros + i;     // acumulador
        i = i+1;                         // incremento de contador

    }

    promNumeros = sumNumeros / 10;

    cout<<"La sumatoria de numeros es: "<<sumNumeros<<endl;
    cout<<"El promedio de los numeros es: "<<promNumeros<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

En la figura anterior, se muestra el código de la solución propuesta, resaltando el uso de la variable "i" que permite recorrer cada número.

Esta variable "i" forma parte de la condición de la Estructura de control repetitiva mientras, debido a que esta variable está siendo modificada (incrementada como contador) dentro de la estructura repetitiva.

Complementa la condición, la cantidad de datos a repetir o recorrer, en este caso diez, pero en otra propuesta de solución, el valor límite puede ser ingresada a través de una variable.

Otro ejemplo de Estructura repetitiva mientras:

Reforzaremos el uso del contador y acumulador, solicitando el ingreso de N calificaciones, para hallar el promedio y la cantidad de calificaciones aprobadas que se ingresaron, además de hallar la calificación mayor (más alta) y la calificación menor (menos alta).

Definimos variables, con un cuadro que le permita definir las tres partes del algoritmo de solución.

Cuadro 10
Solución con partes del algoritmo para Ejemplo3 de una Estructura de control repetitiva mientras

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

N
i
Calif
contAprob
contDsprob
sumCalif
promCalif
mayor
menor
	N
	Calif
		Si(N > 0)
		{ i = 1;
		Mientras (i <= N)
		{
		Leer Calif;
		Si(Calif >=0 and Calif <=20)
		{ Si(Calif >= 11)
		contAprob++
		Sino
		contDsprob++
		Si(i == 1)
		{ mayor = Calif;
		menor = Calif;
		}
		Sino
		Si(Calif > mayor)
		mayor = Calif;
		Sino
		Si(Calif <menor)
		menor = Calif;
		sumCalif = sumCalif + Calif
		i = i+1;
		}
		Sino
		Escribir "ERROR."
		}
		promCalif = sumCalif / N
		Escribir "El promedio de las calificaciones es: "
		Escribir promCalif
		Escribir "La cantidad de aprobados es:"
		Escribir contAprob
		Escribir "La cantidad de desaprobados es:"
		Escribir contDsprob
		Escribir "La calificacion mayor es:"
		Escribir mayor
		Escribir "La calificacion menor es:"
		Escribir menor
		}
		Sino
		Escribir "ERROR. Debe ingresar mayor a cero."
			promCalif
			contAprob
			contDsprob
			mayor
			menor

Fuente: Carol Rojas M.

Solución en Código C/C++:

Figura 65

Solución en Código C/C++ para Ejemplo3 de una Estructura de control repetitiva mientras

```
#include<iostream>
using namespace std;

int main()
{
    int N, i, Calif, mayor, menor;
    int contAprob = 0, contDsprob = 0, sumCalif = 0;
    double promCalif;

    cout<<"Ingrese la cantidad de calificaciones: ";
    cin>>N;

    if(N > 0)
    {
        i = 1;                  // "i" es una variable de recorrdio por calificación

        while (i <= N)
        {   cout<<endl;
            cout<<"Ingrese calificación: ";
            cin>>Calif;

            if(Calif >=0 && Calif <=20)  // valida notas de 0 a 20
            {
                //Hallar aprobados y desaprobados
                if(Calif >= 11)
                   contAprob++;         // contador de aprobados
                else
                    contDsprob++;       // contador de desaprobados

                //Hallar mayor y menor
                if (i == 1)
                    {
                        mayor = Calif;
                        menor = Calif;
                    }
                else
                    if(Calif > mayor)
                        mayor = Calif;
                    else
                        if(Calif <menor)
                            menor = Calif;

                sumCalif = sumCalif + Calif; // acumulador
                i = i+1;                    // contador de calificación
            }
            else
                cout<<"ERROR. Debe ingresar >= 0 y <= 20."<<endl;
        }

        promCalif = sumCalif / N;

        cout<<endl;
        cout<<"El promedio de las calificaciones es:"<<promCalif<<endl;
        cout<<"La cantidad de aprobados es:"<<contAprob<<endl;
        cout<<"La cantidad de desaprobados es:"<<contDsprob<<endl;
        cout<<"La calificación mayor es:"<<mayor<<endl;
        cout<<"La calificación menor es:"<<menor<<endl;
    }
    else
        cout<<"ERROR. Debe ingresar mayor a cero."<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

TEMA N° 2:
ESTRUCTURA DE CONTROL REPETITIVA HACER - MIENTRAS

¿Las estructuras repetitivas permiten mejorar el ingreso de datos?

Existe una estructura repetitiva que permite repetir el ingreso de datos en caso o repetir el algoritmo de solución, en caso lo requiera el usuario.

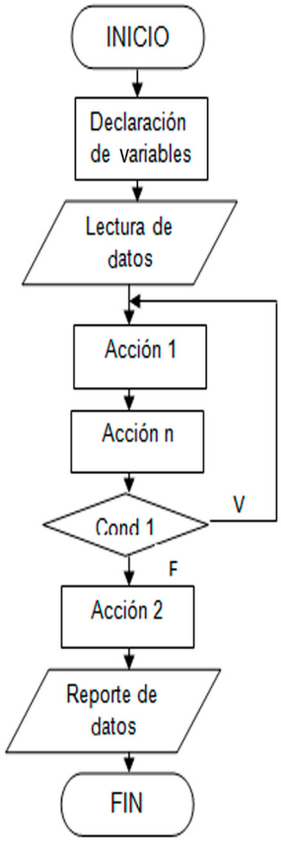
Esta estructura de control es recomendable para repetir el algoritmo una cantidad de veces requerida por el usuario, y no necesariamente asignando un número límite de veces.

Pero considere que también puede aplicar la instrucción break en una estructura repetitiva, para interrumpir abruptamente la repetición del algoritmo.

1. DEFINICIÓN DE ESTRUCTURA DE CONTROL REPETITIVA HACER - MIENTRAS[11]

Es una estructura que realiza al menos una vez un conjunto de acciones, y luego evalúa una expresión condicional, si es verdadero regresa a repetir el conjunto de acciones, si no cumple, sale del bucle.

Figura 66
Diagrama de flujo para una Estructura de control repetitiva hacer - mientras



Fuente: Carol Rojas M.

11 Carrasco, A. (2005). Principios de programación. Algoritmos y su creación en C++. Perú: AC Editores.

Ejemplo de Estructura repetitiva hacer - mientras:

Leer un número entero mayor que cero, sino es así, volver a solicitar el número con esas condiciones.

Solución:

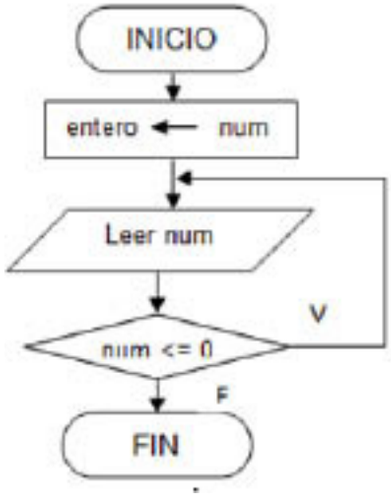
Se sugiere definir la variable que se va a verificar:

Definición de variables:

num: valor del número a validar

Solución en diagrama de flujo:

Figura 67
Solución de Diagrama de flujo para Ejemplo1 de una Estructura de control repetitiva hacer mientras



Fuente: Carol Rojas M.

Solución en Código C/C++:

Figura 68
Solución en Código C/C++
para Ejemplo1 de una Estructura de control repetitiva hacer mientras

```
1   #include<iostream>
2   using namespace std;
3
4   void main( )
5   {    int num;
6
7        do{
8            cout<<" Ingrese un número entero mayor a cero:   ";
9            cin>>num;
10       }while(num <= 0);
11  }
```

Fuente: Carol Rojas M.

Otro ejemplo de Estructura repetitiva hacer - mientras:

Una vendedora de pan tiene cierta cantidad de unidades de pan al comenzar el día. Si cada cliente le pide cierta cantidad de panes, cuantos clientes son atendidos completamente, ¿Cuántos panes quedan para el ultimo cliente?

Solución:

Definimos variables, con un cuadro que le permita definir las tres partes del algoritmo de solución.

Cuadro 11
Solución con partes del algoritmo
para Ejemplo2 de una Estructura de Control repetitiva hacer mientras

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

InventarioPan
cantPan
contClientAtendido
contClientNoAtendido
TotalVendido
rpta
	InventarioPan
	cantPan
	rpta
		Hacer{
		Leer InventarioPan;
		Si(InventarioPan <= 0)
		Escribir "Error, Vuelva ingresar >=0"
		}Mientras(InventarioPan <= 0)
		Hacer{
		Hacer{
		Leer cantPan;
		Si(cantPan <= 0)
		Escribir "Error, Vuelva ingresar >=0"
		}Mientras(cantPan <= 0);
		Si(cantPan <= InventarioPan)
		{
		contClientAtendido = contClientAtendido + 1
		InventarioPan = InventarioPan - cantPan
		TotalVendido = TotalVendido + cantPan
		}
		Sino
		{
		Escribir "No se puede atender esta cantidad."
		contClientNoAtendido = contClientNoAtendido + 1
		}
		Escribir "Desea otra venta de nuevo cliente? "
		Leer rpta;
		}Mientras(rpta == 'S'|| rpta == 's')
		
		Escribir "Clientes No Atendidos:"
		Escribir contClientNoAtendido
		Escribir "Clientes Atendidos:"
		Escribir contClientAtendido
		Escribir "Total Panes Vendidos:"
		Escribir TotalVendido
		Escribir "Queda para ultimo cliente:"
		Escribir InventarioPan
			contClientAtendido
			contClientNoAtendido
			TotalVendido
			InventarioPan

Fuente: Carol Rojas M.

Solución en Código C/C++:

Figura N° 69
Solución en Código C/C++
para Ejemplo2 de una Estructura de control repetitiva hacer mientras

```
#include<iostream>
using namespace std;

int main()
{
    int InventarioPan, cantPan;
    int contClientAtendido = 0, contClientNoAtendido = 0, TotalVendido = 0;
    char rpta;


    // Validar datos con do-while
    do{
        cout<<"Ingrese Total de Panes a vender:\t";
        cin>>InventarioPan;

        if(InventarioPan <= 0)
            cout<<"Error, Vuelva ingresar >=0"<<endl;

    }while(InventarioPan <= 0);             // Si es Vedadero, repite el ingreso


    // Repetir todo el algoritmo de cálculo con do-while
    do{

            // Validar datos con do-while
            do{
                cout<<"Ingrese cantidad de panes por cliente:\t";
                cin>>cantPan;

                if(cantPan <= 0)
                    cout<<"Error, Vuelva ingresar >=0"<<endl;

            }while(cantPan <= 0);     // Si es Vedadero, repite el ingreso


            if(cantPan <= InventarioPan)
            {
                contClientAtendido = contClientAtendido + 1;  // contar clientes atendidos
                InventarioPan = InventarioPan - cantPan;      // decrementar inventario
                TotalVendido = TotalVendido + cantPan;        // acumular ventas

            }
            else
            {
                cout<<"No se puede atender esta cantidad."<<endl;
                contClientNoAtendido = contClientNoAtendido + 1; // contar clientes no atendidos
            }

            cout<<"Desea otra venta de nuevo cliente? (s o n):\t";
            cin>>rpta;

    }while(rpta == 'S'|| rpta == 's');

        cout<<endl;
        cout<<"Clientes No Atendidos:\t\t"<<contClientNoAtendido<<endl;
        cout<<"Clientes Atendidos:\t\t"<<contClientAtendido<<endl;
        cout<<"Total Panes Vendidos:\t\t"<<TotalVendido<<endl;
        cout<<"Queda para ultimo cliente:\t"<<InventarioPan<<endl;


    return 0;
}
```

Fuente: Carol Rojas M.

Conociendo esta nueva estructura de control repetitiva, podemos modificar uno de los programas revisados en la estructura de control selectiva.

Recuerda el siguiente programa:

Figura 70
Programa propuesto a modificar validación de datos

```
#include<string>
using namespace std;

int main()
{
    string nombClie;
    char geneClie;
    double precioProd=100;
    int cantProd;
    double PorcDscto, MontoBruto, MontoDscto, MontoPago;

    cout<<"Ingrese Nombre de Cliente: ";
    cin>>nombClie;

    cout<<"Ingrese Cantidad de Productos: ";
    cin>>cantProd;

    //Validar cantidad mayor a cero
    if(cantProd>0)
    {
        cout<<"Ingrese Genero de Cliente: ";
        cin>>geneClie;

        //Validar que se ingrese genero
        if(geneClie == 'm' || geneClie == 'M' || geneClie == 'f' || geneClie == 'F')
        {
            switch(geneClie)
            {
                case 'm':
                case 'M':
                        {
                            if(cantProd <= 10)
                                PorcDscto = 0.2;
                            else
                                if(cantProd > 10)
                                    PorcDscto = 0.5;
                        }break;

                case 'f':
                case 'F':
                        {
                            if(cantProd <= 10)
                                PorcDscto = 0.2;
                            else
                                if(cantProd > 10)
                                    PorcDscto = 0.4;
                        }break;

            }

            MontoBruto = cantProd * precioProd;
            MontoDscto = MontoBruto * PorcDscto;
            MontoPago = MontoBruto - MontoDscto;

            cout<<"El monto bruto es: ";
            cout<<MontoBruto<<endl;
            cout<<"El monto descuento es: ";
            cout<<MontoDscto<<endl;
            cout<<"El monto Pago es: ";
            cout<<MontoPago<<endl;
        }
        else
            cout<<"ERROR. Debe ingresar femenino o masculino.";
    }
    else
        cout<<"ERROR. Debe ingresar mayor a cero.";

    return 0;
}
```

Fuente: Carol Rojas M.

Si observa la figura anterior, el ingreso de datos de la cantidad de productos y del género, están validados con una estructura de control selectiva (if-else)

La nueva propuesta para cada uno y manteniéndose el resto del programa, sería:

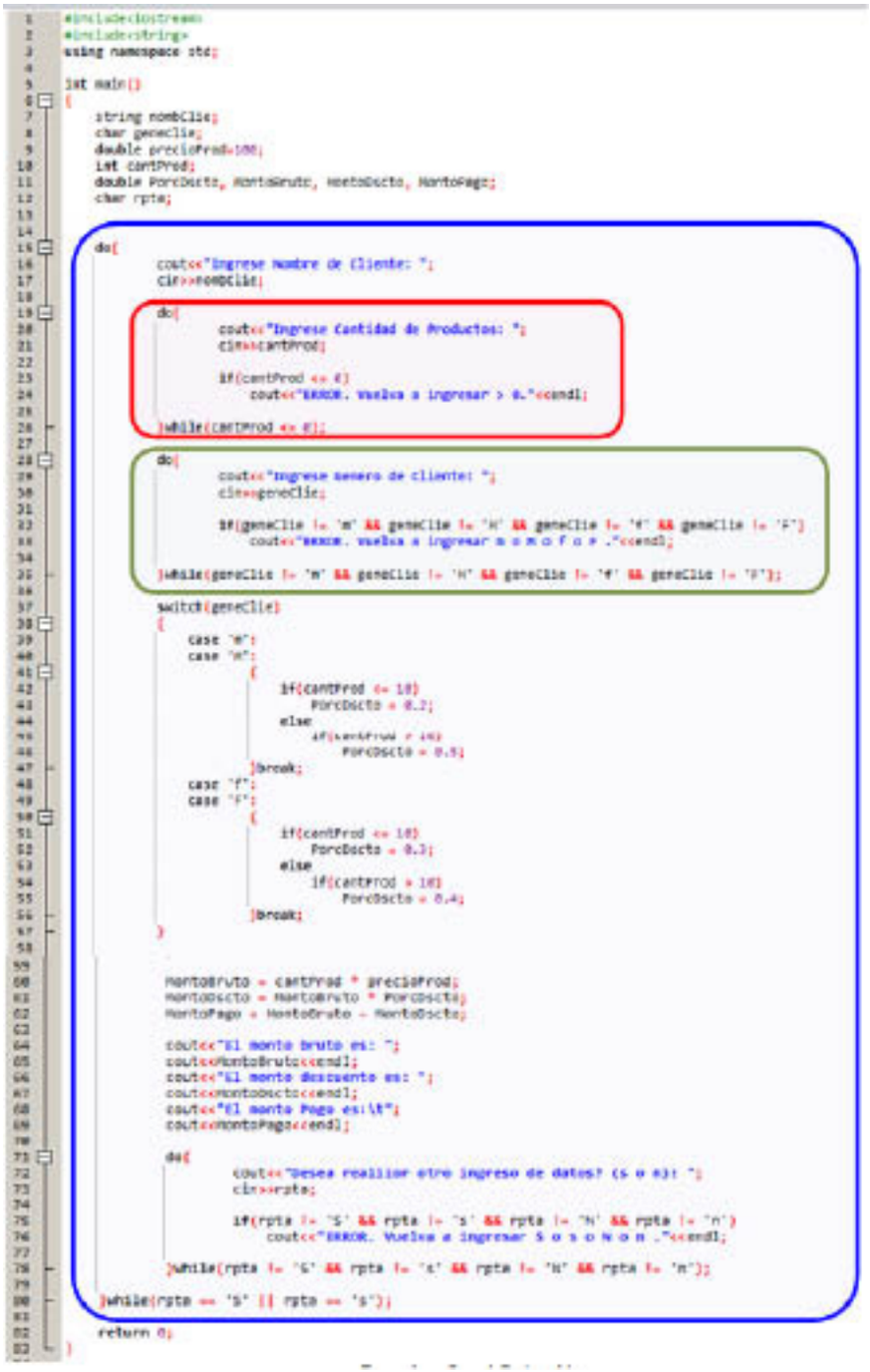
Figura 71
Programa propuesto modificando validación de datos

```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    string nombClie;
    char geneClie;
    double precioProd=100;
    int cantProd;
    double PorcDscto, MontoBruto, MontoDscto, MontoPago;

    cout<<"Ingrese Nombre de Cliente: ";
    cin>>nombClie;

    do{
            cout<<"Ingrese Cantidad de Productos: ";
            cin>>cantProd;

            if(cantProd <= 0)
                cout<<"ERROR. Vuelva a ingresar > 0."<<endl;

    }while(cantProd <= 0);

    do{
            cout<<"Ingrese Género de Cliente: ";
            cin>>geneClie;

            if(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F')
                cout<<"ERROR. Vuelva a ingresar m o M o f o F ."<<endl;

    }while(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F');

    switch(geneClie)
    {
        case 'm':
        case 'M':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.2;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.5;
                    }break;
        case 'f':
        case 'F':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.3;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.4;
                    }break;

    }

    MontoBruto = cantProd * precioProd;
    MontoDscto = MontoBruto * PorcDscto;
    MontoPago = MontoBruto - MontoDscto;

    cout<<"El monto bruto es: ";
    cout<<MontoBruto<<endl;
    cout<<"El monto descuento es: ";
    cout<<MontoDscto<<endl;
    cout<<"El monto Pago es: ";
    cout<<MontoPago<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

Además, aprovechando esta estructura de control repetitiva, podemos iterar o repetir el algoritmo las veces que el usuario del programa lo requiera:

Figura 72
Programa propuesto modificando validación de datos



```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    string nombClie;
    char geneClie;
    double precioProd=100;
    int cantProd;
    double PorcDscto, MontoBruto, MontoDscto, MontoPago;
    char rpta;

    do{
        cout<<"Ingrese Nombre de Cliente: ";
        cin>>nombClie;

        do{
            cout<<"Ingrese Cantidad de Productos: ";
            cin>>cantProd;

            if(cantProd <= 0)
                cout<<"ERROR. Vuelva a ingresar > 0."<<endl;

        }while(cantProd <= 0);

        do{
            cout<<"Ingrese Genero de cliente: ";
            cin>>geneClie;

            if(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F')
                cout<<"ERROR. Vuelva a ingresar m o M o f o F .";<<endl;

        }while(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F');

        switch(geneClie)
        {
            case 'm':
            case 'M':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.2;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.5;
                    }break;
            case 'f':
            case 'F':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.3;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.4;
                    }break;
        }

        MontoBruto = cantProd * precioProd;
        MontoDscto = MontoBruto * PorcDscto;
        MontoPago = MontoBruto - MontoDscto;

        cout<<"El monto bruto es: ";
        cout<<MontoBruto<<endl;
        cout<<"El monto descuento es: ";
        cout<<MontoDscto<<endl;
        cout<<"El monto Pago es:\t";
        cout<<MontoPago<<endl;

        do{
            cout<<"Desea realizar otro ingreso de datos? (S o s): ";
            cin>>rpta;

            if(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n')
                cout<<"ERROR. Vuelva a ingresar S o s o N o n .";<<endl;

        }while(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n');

    }while(rpta == 'S' || rpta == 's');

    return 0;
}
```

Fuente: Carol Rojas M.
SÍNTESIS del TEMA I- II

LECTURA SELECCIONADA Nº 1:

ESTRUCTURAS DE REPETICIÓN[4]

Son aquellas que nos permiten variar o alterar la secuencia normal de ejecución de un programa haciendo posible que un bloque de instrucciones se ejecute más de una vez de forma consecutiva.

Estructura mientras.

La estructura mientras se caracteriza porque su diseño permite repetir un bloque de instrucciones de 0 – n veces, es decir, que en aquellos casos en los que la condición establecida sea verdadera, el número de veces que se ejecutará dicho bloque de instrucciones será una vez como mínimo y n veces como máximo, mientras que en el caso de que la condición establecida sea falsa dicho bloque de instrucciones no se ejecutará ninguna vez. La forma general de esta estructura es:

While (expresión condicional)

{
Sentencia1
sentenciaN
}

Un ejemplo con esta estructura es el siguiente:

Diseño del algoritmo correspondiente a un programa que lee un número entero positivo y determina el número de dígitos decimales necesarios para la representación de dicho valor.

VARIABLES:

Ndigitos Numérico entero
Pot Numérico entero
N Numérico entero
ALGORITMO:
Ndigitos = 1
Pot =10 Leer N
Mientras Pot <= N
Ndigitos ++
Pot*=10
Fin mientras
Escribir "Se necesitan ", Ndigitos
Estructura repetir – mientras.

La estructura repetir – mientras se caracteriza porque su diseño permite repetir un bloque de instrucciones de 1 – n veces, es decir, ya sea verdadera o falsa la condición establecida, el número de veces que se ejecutará el bloque de instrucciones será de una vez como mínimo y de n veces como máximo.

La forma general de esta estructura es:

do
{
sentencia1
sentenciaN
}

Un ejemplo con este tipo de estructuras es el siguiente:

Algoritmo correspondiente a un programa que lee un número entero positivo y seguidamente escribe el carácter asterisco (*) un número de veces igual al valor numérico leído. En aquellos casos, en los que el valor leído no sea entero, numérico y positivo se deberá escribir solamente un asterisco.

VARIABLES:

ast Numérico entero
numast Numérico entero

ALGORITMO:
ast =0
Leer numast

Repetir
ast++
escribir '*'
Mientras(ast<numast)

12 Alcaraz, R. (2009). Una visión práctica de la Programación en C. Méxicohttps://ruidera.uclm.es/xmlui/bitstream/handle/10578/43/pro-

ACTIVIDAD FORMATIVA N° 1

Elabora el programa de cómputo para cada situación propuesta.

INSTRUCCIONES:

1. Lee y analiza el tema N° 1 y N° 2 extrae las ideas fundamentales de la estructura de control repetitiva para la programación.
2. Observe el vídeo **"Tutorial C++ 13. While y Do While"** y complemente la información obtenida en el tema N°1.
3. Observe el vídeo **"Tutorial Programación C++ -Clase 7 bucles - do while"** y complemente la información obtenida en el tema N° 2.
4. Elabore un programa en lenguaje C/C++ para las siguientes situación problema:

4.1. Suponga que tiene la calificación final de un grupo de alumnos, calcular la calificación promedio y la calificación más alta y la más baja de todo el grupo.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

4.2. Elabore el código para una tienda para leer por cada cliente el monto total de su compra, al final del día reportar el monto total acumulado de ventas y el número de clientes atendidos.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

gramacion_en_C.pdf?sequence=1

TEMA N° 3:
ESTRUCTURA DE CONTROL REPETITIVA DESDE/POR

Estimado estudiante, hasta este punto se ha presentado la repetición del algoritmo, mientras el usuario acepte o no continuar con el proceso.

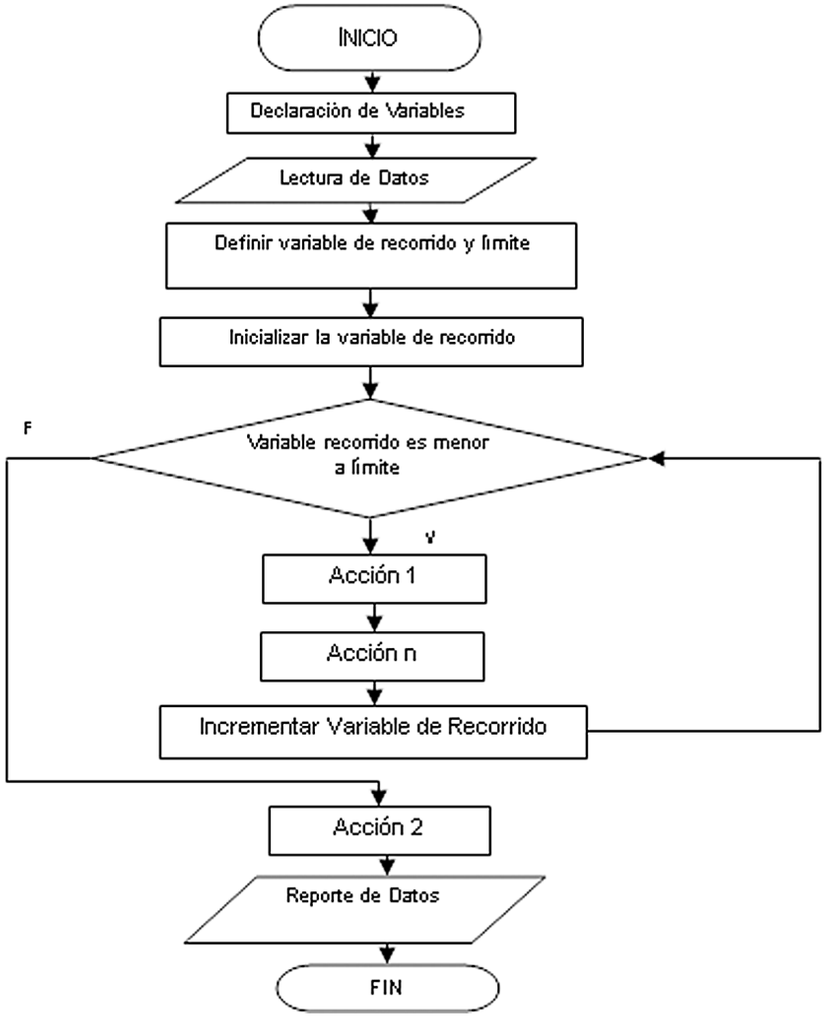
Sin embargo, cuando se conoce la cantidad exacta de repeticiones se hace uso de otra estructura de control de repetición.

1. DEFINICIÓN DE ESTRUCTURA DE REPETITIVA DESDE/POR

Esta estructura de control permite repetir un conjunto de acciones, hasta llegar a un límite dado, utilizando una variable de recorrido por cada repetición.

Generalmente, esta variable de recorrido se define con la letra "i" (índice de recorrido), pero puede usarse otra variable y que puede inicializar en cero o en cualquier otro valor, para luego incrementarse o decrementarse.

Figura 73
Diagrama de flujo para una Estructura de control repetitiva desde/por



Fuente: Carol Rojas M.

Ejemplo de Estructura repetitiva desde/por:

Tenemos el requerimiento de calcular el factorial de un número.

Solución:

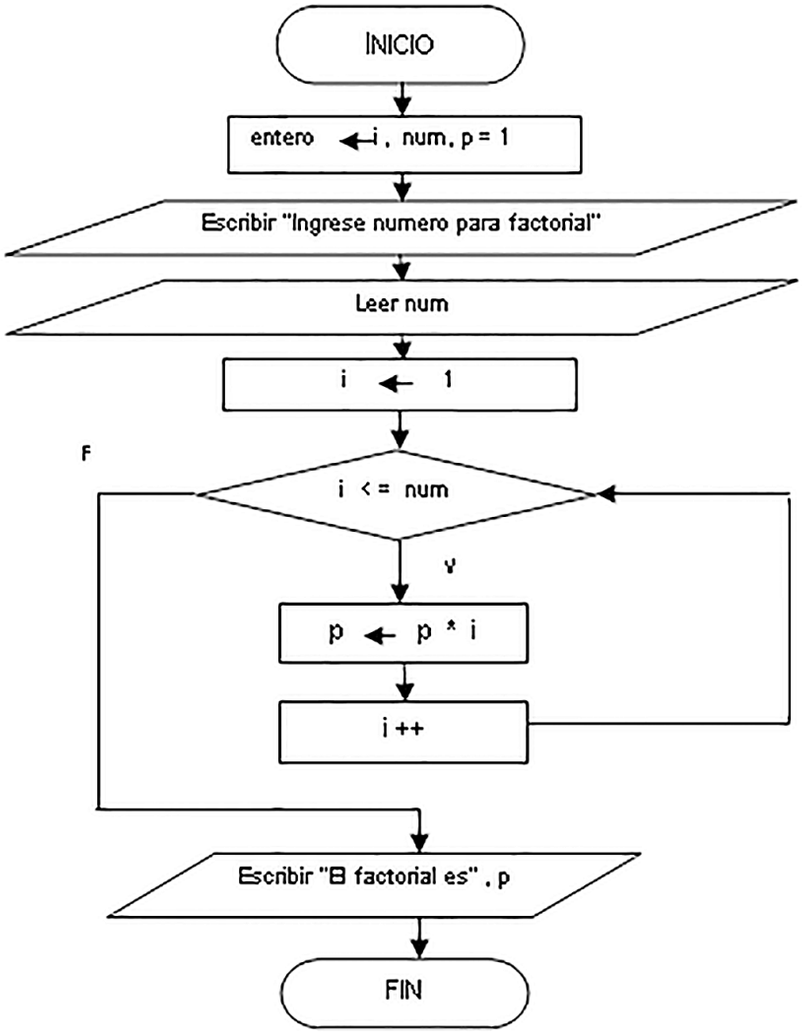
Se sugiere como variable el ingreso de un número entero positivo:

Definición de variables:

num: valor del número entero positivo

Solución en Diagrama de flujo:

Figura 74
Solución de Diagrama de flujo
para Ejemplo1 de una Estructura de control repetitiva desde/por



Fuente: Carol Rojas M.

Solución en Código C/C++:

Figura 75
Solución en Código C/C++
para Ejemplo1 de una Estructura de control repetitiva desde/por

```
#include<iostream>
using namespace std;

void main( )
{
    int num, i, p=1;

    cout<<"Ingrese numero para factorial: ";
    cin>>num;

    for(i=1;i<=num; i++)
    {
        p = p * i;
    }

    cout<<"El factorial de "<<n<<"es: "<<p;
}
```

Fuente: Carol Rojas M.

En la figura anterior, se puede decir, que para hallar el factorial de un número, como todos sabemos, es la multiplicación del número disminuido en uno.

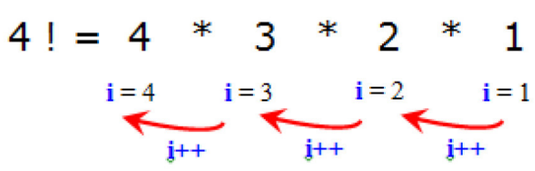
Ejemplo:

$$4! = 4 * 3 * 2 * 1$$

Pero recuerde que no se puede disminuir hasta cero, sino la multiplicación se convierte en cero, no permitiendo hallar el valor del factorial, esa es la razón que el producto se inicializa en uno (p = 1) para no afectar la multiplicación.

Para aplicar esta estructura repetitiva, al ejemplo anterior del factorial de un número, en lugar de analizarlo como la disminución del número, lo planteamos como el incremento (i++) desde el uno (i=1) hasta llegar al número (i<=num):

Ejemplo:



Finalmente, cada valor actualizado de la variable "i" se multiplica con el valor anterior, dando como resultado el producto o el valor del factorial del número.

Recuerde que esta estructura repetitiva requiere tener un valor máximo o límite hasta donde llegará la variable de recorrido al ser incrementada.

Por lo que, primero se inicializa la variable de recorrido (i), luego se verifica que no haya superado el límite, si esto es así se realizan las instrucciones (para este ejemplo, la multiplicación), finalmente se incrementa la variable de recorrido (i++) y se repite el proceso inicializando nuevamente la variable de recorrido, después del incremento.

Un ejemplo más de aplicación de esta estructura repetitiva:

Elaborar un programa que le a un número n> 0 e imprima los n términos de la serie. Además, debe imprimir la suma de los n términos:

$$\frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \frac{4}{5} + \ldots + \frac{n}{n+1}$$

Solución:

Definimos variables, con un cuadro que le permita definir las tres partes del algoritmo de solución.

Cuadro 12
Solución con partes del algoritmo
para Ejemplo2 de una Estructura de control repetitiva desde/por

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

n
i
denominador
suma
	n	Hacer{
		Leer n;
		Si(n<= 0)
		
		Escribir "Error, Vuelva ingresar >=0"
		}Mientras(n <= 0)
		Desde(i = 1; i <= n; i++)
		{
		Escribir i, "/", i + 1
		denominador = i + 1
		suma = suma + (i / denominador)
		}
		
		Escribir " Suma es: "
		Escribir suma
			suma

Fuente: Carol Rojas M.

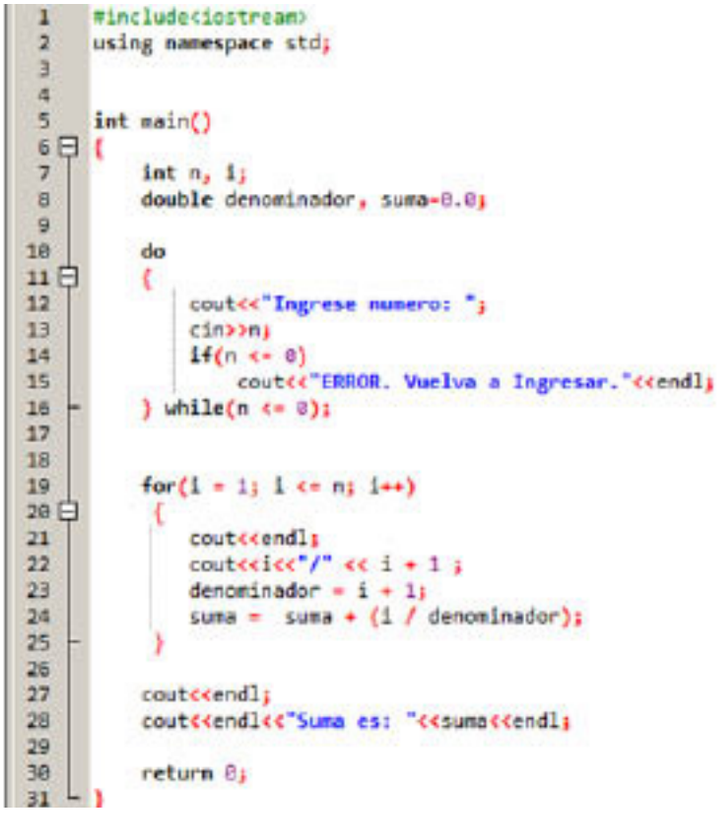
En este caso, la variable "i" inicializa en uno, como indica el valor en la serie de números, y va incremento su valor hasta llegar a ser igual "n" que es el número natural ingresado como límite.

Luego, se muestra en pantalla la serie n / (n+1), la cual es recorrido desde uno con la variable "i".

Para ordenar y mejorar la manipulación de valores, antes de realizar la acumulación o sumatoria, se usa una variable denominador para n+1, y de esta manera facilitar el cálculo.

Solución en Código C/C++:

Figura 76
Solución en Código C/C++
para Ejemplo2 de una Estructura de control repetitiva desde/por



```
#include<iostream>
using namespace std;


int main()
{
    int n, i;
    double denominador, suma=0.0;

    do
    {
        cout<<"Ingrese numero: ";
        cin>>n;
        if(n <= 0)
            cout<<"ERROR. Vuelva a Ingresar."<<endl;
    } while(n <= 0);


    for(i = 1; i <= n; i++)
     {
        cout<<endl;
        cout<<i<<"/" << i + 1 ;
        denominador = i + 1;
        suma =  suma + (i / denominador);
     }

    cout<<endl;
    cout<<endl<<"Suma es: "<<suma<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

Otro ejemplo de uso de esta estructura repetitiva:

Elaborar un programa que permita realizar hasta n veces (intentos) las operaciones de un cajero automático: depositar, retirar y ver saldo.

Hipotéticamente, considere que se tiene una cierta cantidad de oportunidades o intentos para hacer uso del cajero automático, ya que esta restricción será mejorada si aplica la estructura hacer – mientras, para solicitar al usuario si requiere o no realizar una operación.

Solución:

Definimos variables, con un cuadro que le permita definir las tres partes del algoritmo de solución.

Cuadro 13
Solución con partes del algoritmo
para Ejemplo3 de una Estructura de control repetitiva desde/por

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

CantVeces
op
i
mdp
mrt
msd
	CantVeces
	op
		Hacer{
		Leer CantVeces;
		Si(CantVeces <= 0)
		Escribir "ERROR. Vuelva a ingresar."
		}Mientras(CantVeces <= 0)
		for(i=1; i <= CantVeces; i++)
		{ Escribir "==== CAJERO ===";
		Escribir "1. Depositar"
		Escribir "2. Retirar"
		Escribir "3. Saldo"
		Leer op;
		Si(op >= 1 && op <=3)
		{
		Si(op)
		{ caso 1: {
		Hacer{
		Leer mdp;
		Si(mdp<=0)
		Escribir "ERROR."
		}Mientras(mdp<=0)
		msd= msd + mdp;
		}break;
		caso 2: {
		Hacer{
		Leer mrt;
		Si(mrt<=0)
		Error "ERROR."
		}Mientras(mrt<=0)
		Si(mrt <= msd)
		msd= msd - mrt;
		Sino
		Escribir "No tiene saldo suficiente."
		}break;
		case 3:{
		Escribir "Su saldo es: "
		Escribir msd
		}break;
		}
		Escribir "Le quedan ", CantVeces-i," intentos"
		}
		Sino
		{ Escribir "ERROR. Opción no válida."
		Escribir "Se agotan los intentos."
		}
		}
			msd

Fuente: Carol Rojas M.

Solución en Código C/C++:

Figura 77
Solución en Código C/C++
para Ejemplo3 de una Estructura de control repetitiva desde/por

```
#include<iostream>
using namespace std;

int main()
{
    int CantVeces, op, i;
    float mdp, mrt, msd = 0;

    do{
            cout<<"Ingrese Cantidad de Intentos: ";
            cin>>CantVeces;
            if(CantVeces <= 0)
                cout<<"ERROR. Vuelva a Ingresar."<<endl;
    }while(CantVeces <= 0);

    for(i=1; i <= CantVeces; i++)
    {       cout<<"==== CAJERO ===";
            cout<<"\n";
            cout<<"1. Depositar"<<endl;
            cout<<"2. Retirar"<<endl;
            cout<<"3. Saldo"<<endl;
            cout<<"Seleccione opcion: ";
            cin>>op;
            if(op >= 1 && op <=3)
                {
                    switch(op)
                        {   case 1: {
                                        do{
                                            cout<<"Ingrese monto a depositar: ";
                                            cin>>mdp;
                                            if(mdp<=0)
                                                cout<<"ERROR. Vuelva a Ingresar."<<endl;
                                        }while(mdp<=0);

                                        msd= msd + mdp;
                                    }break;

                                case 2: {
                                        do{
                                            cout<<endl<<"Ingrese monto a retirar: ";
                                            cin>>mrt;
                                            if(mrt<=0)
                                                cout<<endl<<"ERROR. Vuelva a Ingresar."<<endl;
                                        }while(mrt<=0);

                                        if(mrt <= msd)
                                            msd= msd - mrt;
                                        else
                                            cout<<endl<<"No tiene saldo suficiente."<<endl;
                                    }break;

                                case 3:{
                                        cout<<endl<<"Su saldo es: "<<msd<<endl;
                                    }break;
                        }

                    cout<<endl<<"le quedan "<<CantVeces-i<<" intentos"<<endl;
                }
            else
                {   cout<<endl<<"ERROR. Opcion no valida."<<endl;
                    cout<<"Se agotan los intentos."<<endl;
                }
    }
    return 0;
}
```

Fuente: Carol Rojas M.

SÍNTESIS del TEMA III

PROGRAMACIÓN ESTRUCTURADA

Conjunto de técnicas para elaborar y depurar programas.

Requiere de un programa traductor, para convertir a código máquina lo escrito en el lenguaje de programación.

Traductores:

- Intérprete
- Compilador

ACTIVIDAD FORMATIVA N° 2

Elabora el programa de cómputo para cada situación propuesta.

INSTRUCCIONES:

1. Lee y analiza el tema N° 3 y extrae las ideas fundamentales de la estructura de control repetitivas para la programación.
2. Observe el vídeo "Tutorial Programación C++ -Clase 7 bucles – for" y complemente la información obtenida en el tema N°3.
3. Elabore un programa en lenguaje C/C++ para las siguientes situación problema:

3.1 En una universidad se pueden llevar hasta 22 créditos en un ciclo. Elabora el código que permita a un alumno matricularse sin pasarse del límite. Validando los datos de entrada y solicitando si desea seguir matriculándose en más créditos por cada curso.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

El instrumento para calificar es una lista de cotejo con los siguientes criterios, para un archivo Word con la tabla de las partes del algoritmo, y el archivo fuente (.cpp) de cada ejercicio, en una carpeta ApellidoNombreAlumno.zip:

CRITERIO	PUNTAJE
Elaboración de la tabla con las tres partes del algoritmo y las variables a usar.	2
Declarar las variables con su respectivo tipo de dato.	2
Ingresar la cantidad de crédito por curso, validando con la sentencia repetitiva hacer-mientras.	4
Verificar si no excede el máximo de crédito a matricular.	3
Realizar los cálculos para matricular un curso, según su crédito sin exceder el máximo de créditos.	2
Repetir el algoritmo de matricular un curso según su crédito, con la sentencia repetitiva hacer-mientras.	4
Usar contador y acumulador para mostrar la cantidad de cursos matriculados y el total de créditos acumulados.	3

GLOSARIO DE LA UNIDAD III

B

BUCLE.
Repetición, iteración de un determinado proceso o sentencias de programación.

C

CLIENTE DE PROGRAMA.
Persona beneficiada con la elaboración del programa, por quien se expresan los requerimientos.

E

EXPRESIÓN CONDICIONAL.
Son sentencias con operadores de conjunción o disyunción, para permitir evaluar la realización de un algoritmo.

L

LÍMITE.
Valor máximo a ser recorrido, en una estructura repetitiva.

O

OPERACIÓN.
Instrucción o sentencia de programación a realizar en el programa.

R

REPOSITORIO.
Almacén o espacio de memoria para albergar un conjunto de datos.

U

USUARIO DE PROGRAMA.
Persona que hará uso de las especificaciones del programa.

V

VARIABLE DE RECORRIDO.
Espacio de memoria que se inicializa con un valor y puede ser modificado al incrementarse o decrementarse.

BIBLIOGRAFIA DE LA UNIDAD III

- **Carrasco, A. (2005).** *Principios de programación*. Algoritmos y su creación en C++. Perú: AC Editores.
- **Joyanes, L. (2008).** *Fundamentos de programación.* Madrid: McGRAW-HILL.

AUTOEVALUACION Nº 3

1. Indique el flujo de trabajo que realiza la estructura de control repetitiva Mientras.

   a) Primero realiza las acciones y luego evaluar la condición para repetir.

   b) Primero inicializa la variable de recorrido y luego realiza las acciones.

   c) Primero realiza el incremento de la variable y luego realiza las acciones.

   d) Primero realiza las acciones y luego realiza el incremento de la variable.

   e) Primero evalúa la condición, luego realiza las acciones a repetir.

2. Dado el siguiente bloque de código que utiliza una estructura repetitiva, indique que valores lógicos de la tabla de verdad (Verdadero=V y Falso=F) se necesita en la condición, para que repita el algoritmo de sumar:

```
while(i > 0 && suma <= 10)
{       i++;
        suma = suma + i;
}
```

   a) V,V

   b) F,F

   c) F,V

   d) V,F

   e) F,F,V

3. Indique el flujo de trabajo que realiza la estructura de control repetitiva Hacer - Mientras.

   a) Primero inicializa la variable de recorrido y luego realiza las acciones.

   b) Primero realiza el incremento de la variable y luego realiza las acciones.

   c) Primero realiza las acciones y luego evaluar la condición para repetir.

   d) Primero evalúa la condición, luego realiza las acciones a repetir.

   e) Primero realiza las acciones y luego realiza el incremento de la variable.

4. Dado el siguiente bloque de código que utiliza una estructura repetitiva, indique que valores lógicos de la tabla de verdad (Verdadero=V y Falso=F) se necesita en la condición, para que no solicite nuevamente el ingreso el valor:

```
do{
        cout<<"Ingrese dato: ";
cin>>valor;
}while(valor=='S' || valor=='s');
```

a) V,V

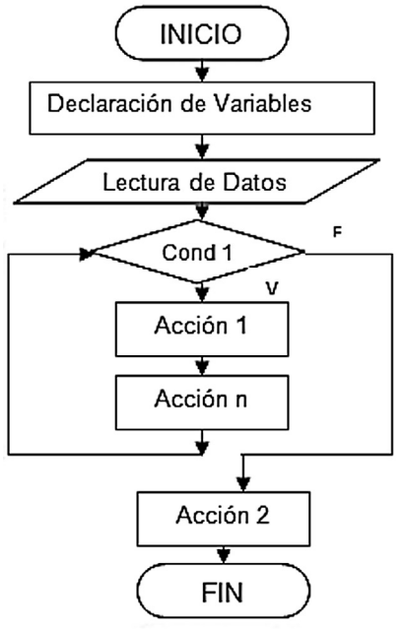
b) F,V

c) V,F

d) F,F

e) F,F,V

5. Indique a que estructura de control pertenece la siguiente figura:



a) Selectiva anidada

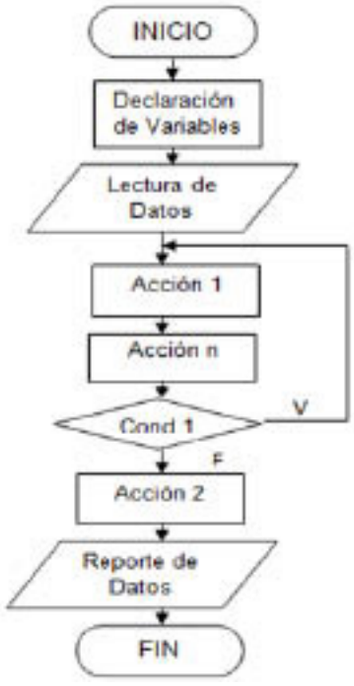
b) Selectiva simple

c) Repetitiva mientras

d) Repetitiva hacer-mientras

e) Selectiva compuesta

6. Indique a que estructura de control pertenece la siguiente figura:



a) Selectiva compuesta

b) Repetitiva hacer-mientras

c) Selectiva anidada

d) Selectiva simple

e) Repetitiva mientras

7. La instrucción "break", además de la estructura de control selectiva múltiple, indique en que otra estructura de control se puede usar.

a) Selectiva simple

b) Repetitivas

c) Selectiva compuesta

d) Secuencial

e) Selectiva anidada

8. En la estructura repetitiva desde/por, se usa la variable de recorrido tiene las siguientes características:

a) Se define como "i" y no se puede usar otra letra.

b) Obligatoriamente, se inicializa con el valor cero.

c) Sólo puede incrementar su valor para volver a iniciar.

d) Se define como "i" o cualquier otra letra con cualquier valor.

e) Solo puede decrementar su valor para volver a iniciar.

9. Indique la alternativa que presenta el orden de ejecución de la estructura repetitiva desde/por:

```
for( i=0; i<n; i++)
      {   acción 1;
          acción m;
       }
```

a) Inicializa la variable de recorrido, compara que esté dentro del límite, incrementa la variable, realiza las acciones e inicia nuevamente el mismo orden de ejecución.

b) Inicializa la variable de recorrido, compara que esté dentro del límite, incrementa la variable, realiza las acciones y finaliza el orden de ejecución.

c) Inicializa la variable de recorrido, compara que esté dentro del límite, realiza las acciones, incrementa la variable e inicia nuevamente el mismo orden de ejecución.

d) Inicializa la variable de recorrido, incrementa la variable, compara que esté dentro del límite, realiza las acciones e inicia nuevamente el mismo orden de ejecución.

e) Inicializa la variable de recorrido, decrementa la variable, compara que esté dentro del límite, realiza las acciones e inicia nuevamente el mismo orden de ejecución.

10. En la estructura repetitiva desde/por, si se quisiese recorrer hasta el límite LIM = 10, y la variable de recorrido "i" se inicializa con el valor cero, indique la condición que debería escribirse.

a) for(i = 0; i <= LIM; i++).

b) for(i = 0; i < LIM-1; i++).

c) for(i = 0; i <= LIM-2; i++).

d) for(i = 0; i < LIM-2; i++).

e) for(i = 0; i < LIM; i++).

UNIDAD IV

"MÓDULOS PARA LA PROGRAMACIÓN: FUNCIONES Y PROCEDIMIENTOS"

DIAGRAMA DE PRESENTACIÓN DE LA UNIDAD IV

- Al finalizar la unidad, el estudiante será capaz de elaborar programas computacionales, aplicando los módulos de programación: funciones y procedimientos.

CONTENIDOS	ACTIVIDADES FORMATIVAS (HABILIDADES Y ACTITUDES)	SISTEMA DE EVALUACIÓN (TÉCNICAS Y CRITERIOS)
Tema Nº 1: Módulos de programas: 1 Definición de módulo de programa 2 Paso de parámetros 2 Tipos de módulos: función y procedimiento **Tema Nº 2: Librerías de programación:** 1 Definición de librería de programación **Tema Nº 3: Funciones recursivas:** 1 Definición de función recursiva	• Lee y analiza el tema Módulos de Programa. Observa el vídeo Función y procedimiento en C++ y elabora un programa en lenguaje C/C++ por cada situación propuesta considerando la lectura N° 1. • Lee y analiza el tema de librerías de programación. Observa el vídeo Tutorial 15 de C++ - Headers (Como crear librerías), elabora un programa en lenguaje C/C++ por cada situación propuesta. • Lee y analiza el tema de funciones recursivas. Observa el vídeo Función Fibonacci recursiva. Ejemplo, y los compara elaborando el módulo de otros algoritmos recursivos.	**Procedimientos e indicadores de evaluación permanente** • Entrega puntual de trabajos realizados. • Calidad, coherencia y pertinencia de contenidos desarrollados. • Prueba teórico-práctica, individual. • Actividades desarrolladas en sesiones tutorizadas. **Criterios de evaluación para diagrama de representación:** • Cuadro de identificación de partes del algoritmo: entrada, proceso, salida. • Programas elaborados con estructuras de control de la programación, módulos y librerías en un lenguaje de programación propuesto, considerando las partes del algoritmo.

RECURSOS:

VIDEOS:

Tema Nº 1 (desde minuto 1:06 hasta 3:40)

Función y procedimiento en C++. Ubicado en:

https://www.youtube.com/watch?v=dSN2O-J8YUg

Tema Nº 2 (desde minuto 2:05 hasta 5:48)

Tutorial 15 de C++ - Headers (Como crear librerías) . Ubicado en:

https://www.youtube.com/watch?v=BdnaU1c5TRc

Tema Nº 3 (hasta 3:02)

Función Fibonacci recursiva. Ejemplo. Ubicado en:

https://www.youtube.com/watch?v=MEUutt1vQ8E

DIAPOSITIVAS ELABORADAS POR EL DOCENTE:

Lectura complementaria:

Lectura seleccionada N° 1

Joyanes, L. (2008). Introducción a los subalgoritmos o subprogramas. Fundamentos de programación. 4a Edición. España : McGRAW-HILL.

INSTRUMENTO DE EVALUACIÓN	• Prueba mixta N° 2
BIBLIOGRAFÍA (BÁSICA Y COMPLEMENTARIA)	**Básica** **Joyanes, L. (2008)**. Introducción a los subalgoritmos o subprogramas. Fundamentos de Programación. 4a Edición. España : McGRAW-HILL. **Complementaria** **Raffo, E. (2000).** Turbo C++. Lima: Mundigraph.
RECURSOS EDUCATIVOS DIGITALES	**Leiva, J.** Procedimientos y funciones. (20 de Junio de 2015). Ubicado en: http://www.lcc.uma.es/~jlleivao/introduccion/parte3.pdf El juego de los discos: La Torre de Hanoi (20 de junio de 2015). Ubicado en: http://www.uterra.com/juegos/torre_hanoi.php

TEMA N° 1:
MÓDULOS DE PROGRAMAS

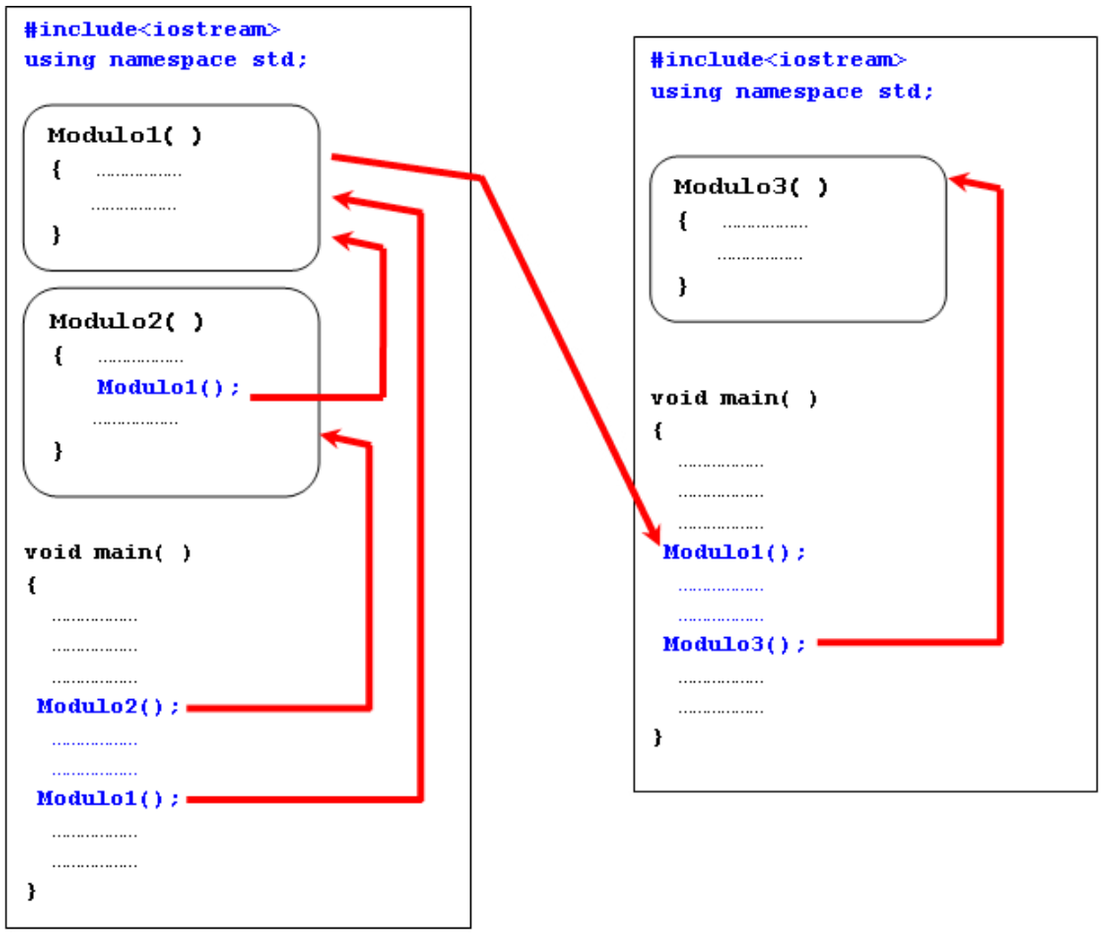
Estimado estudiante, hasta esta sección ha elaborado programas que, según el requerimiento suelen ingresar valores de similares tipos de datos, o aplican algoritmos similares en diferentes situaciones de la realidad.

En programación, se ofrece una estrategia para mejorar la reutilización de algoritmos en diferentes programas, buscando la eficiencia en la elaboración de código.

1. DEFINICIÓN DE MÓDULO DE PROGRAMA

El módulo de programa es un segmento o porción de código, independiente y reutilizable, para el mismo programa o para otros programas.

Figura 78
Ejemplo de módulo de programa



Fuente: Carol Rojas M.

2. PASO DE PARÁMETROS

Es la forma de enviar los valores al invocar a un bloque de código (módulo) en el mismo programa o en otro.

Se conocen dos formas de pasar parámetros:

2.1 PASO DE PARÁMETROS POR VALOR (PARÁMETROS DE ENTRADA)

Guarda en memoria una copia temporal de la variable, y en el módulo solo se utiliza la copia, cuando se modifica el valor del parámetro solo afecta al almacenamiento temporal, la variable actual fuera del procedimiento nunca se modifica.

2.2 PASO DE PARÁMETROS POR REFERENCIA (PARÁMETROS DE ENTRADA/SALIDA)

Los cambios que se efectúen sobre una variable dentro del módulo se mantienen incluso después de que este haya terminado, es decir los cambios afectan al programa principal, y que se usa la variable a través de su dirección de memoria (usa el operador de dirección &).

Figura 79
Ejemplo de paso de parámetros

```
#include<iostream>
using namespace std;

void moduloPrueba(int a, int &b)
{
    a=a+2;
    b=b+2;

    cout<<"Valor de X cambiado en este modulo:\t\t\t";
    cout<<a;                //Muestra el valor cambiado
    cout<<"\n";
    cout<<"Valor de Y cambiado en este modulo:\t\t\t";
    cout<<b;                //Muestra el valor de cambiado
    cout<<"\n";
}

int main()
{
    int x, y;

    x=1;
    y=1;

    moduloPrueba(x,y);      // invoca al modulo

    cout<<"\n";
    cout<<"Valor de X sin cambiar --Paso por Valor: ";
    cout<<x;               //Muestra el valor de "x" sin cambiar---Paso por Valor de variable

    cout<<"\n";
    cout<<"Valor de Y cambiado --Paso por Referencia: ";
    cout<<y;              //Muestra el valor de "y" cambiado en la funcion--Paso por Referencia

    cout<<"\n\n";

    return 0;
}
```

```
F:\USB\1Algoritmia y Estructura de datos\Programas\Paso...
Valor de X cambiado en este modulo:     3
Valor de Y cambiado en este modulo:     3

Valor de X sin cambiar --Paso por Valor: 1
Valor de Y cambiado --Paso por Referencia: 3

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Fuente: Carol Rojas M.

3. TIPOS DE MÓDULOS

Existen dos tipos de módulos que pueden usar los dos tipos de pasos de parámetros o no, es decir, pueden tener o no tener argumentos.

3.1 PROCEDIMIENTO

Son módulos que realizan acciones de programa, pero no devuelve valor.

Declaración sin argumentos:

void NombreProcedimiento ( )

Declaración con argumentos:

void NombreProcedimiento (argumentos)

3.2 FUNCIÓN

Son módulos que realizan acciones de programa, y devuelve 01 valor.

Declaración sin argumentos:

Tipo_dato NombreFunción ( )

Declaración con argumentos:

Tipo_dato NombreFunción (argumentos)

EJEMPLO:

Considere la siguiente figura con un programa simple para sumar dos números:

Figura 80
Ejemplo de programa simple para sumar dos números

```
1  #include<iostream>
2  using namespace std;
3
4  void main()
5  {
6      int a,b,c;
7
8      cout<<"Ingrese valor de a: \t";
9      cin>>a;
10
11     cout<<"Ingrese valor de b: \t";
12     cin>>b;
13
14     c=a+b;
15     cout<<c;
16     cout<<"\n";
17 }
```

Fuente: Carol Rojas M.

El mismo programa puede ser elaborado con procedimientos, con o sin argumentos:

Figura 81
Ejemplo de programa para sumar dos números: Procedimiento con argumentos

```
1  //Procedimientos con Argumentos:
2
3  #include<iostream>
4  using namespace std;
5
6  void sumar(int a, int b)
7  {
8      int c;
9      c=a+b;
10     cout<<c;
11     cout<<"\n";
12 }
13
14 void main()
15 {
16     int a,b;
17     cout<<"Ingrese valor de a: \t";
18     cin>>a;
19
20     cout<<"Ingrese valor de b: \t";
21     cin>>b;
22
23     sumar(a,b);
24 }
```

Fuente: Carol Rojas M.

Figura 82
Ejemplo de programa para sumar dos números: Procedimiento sin argumentos

```
1  //Procedimientos sin Argumentos:
2
3  #include<iostream>
4  using namespace std;
5
6  void sumar()
7  {
8      int a,b,c;
9      cout<<"Ingrese valor de a: \t";
10     cin>>a;
11
12     cout<<"Ingrese valor de b: \t";
13     cin>>b;
14
15     c=a+b;
16     cout<<c;
17 }
18
19 void main()
20 {
21     sumar( );
22 }
```

Fuente: Carol Rojas M.

También el mismo programa puede ser elaborado con funciones, con o sin argumentos:

Figura 83
Ejemplo de programa para sumar dos números: Función con argumentos

```
1  //Funciones con Argumentos:
2
3  #include<iostream>
4  using namespace std;
5
6  int sumar(int a, int b)
7  {
8      int c;
9      c=a+b;
10
11     return c;
12 }
13
14 void main()
15 {
16     int sum,a,b;
17
18     cout<<"Ingrese valor de a: \t";
19     cin>>a;
20
21     cout<<"Ingrese valor de b: \t";
22     cin>>b;
23
24     sum=sumar(a,b);
25
26     cout<<"Valor de la suma de a y b:  "<<sum;
27     cout<<"\n";
28 }
```

Fuente: Carol Rojas M.

Figura 84
Ejemplo de programa para sumar dos números: Función sin argumentos

```
1  //Funciones sin Argumentos:
2
3  #include<iostream>
4  using namespace std;
5
6  int sumar()
7  {
8      int a,b,c;
9
10     cout<<"Ingrese valor de a: \t";
11     cin>>a;
12
13     cout<<"Ingrese valor de b: \t";
14     cin>>b;
15
16     c=a+b;
17
18     return c;
19 }
20
21 void main()
22 {
23     int sum;
24
25     sum=sumar();
26
27     cout<<"Valor de la suma de a y b:  "<<sum;
28     cout<<"\n";
29 }
```

Fuente: Carol Rojas M.

Existe otra manera de definir un procedimiento y una función, declarándola como una variable global, y no es necesario tener un orden de creación en el módulo principal:

Figura 85
Ejemplo de programa para sumar dos números: Otra forma de procedimiento con argumentos

```
1  //Procedimientos con Argumentos
2
3  #include<iostream>
4  using namespace std;
5
6  void sumar(int a, int b);
7
8  int main()
9  {
10     int a,b;
11
12     cout<<"Ingrese valor de a: \t";
13     cin>>a;
14
15     cout<<"Ingrese valor de b: \t";
16     cin>>b;
17
18     sumar(a,b);
19
20     return 0;
21 }
22
23 void sumar(int a, int b)
24 {
25     int c;
26
27     c=a+b;
28
29     cout<<c;
30     cout<<"\n";
31 }
```

Fuente: Carol Rojas M.

Figura 86
Ejemplo de programa para sumar dos números: Otra forma de función con argumentos

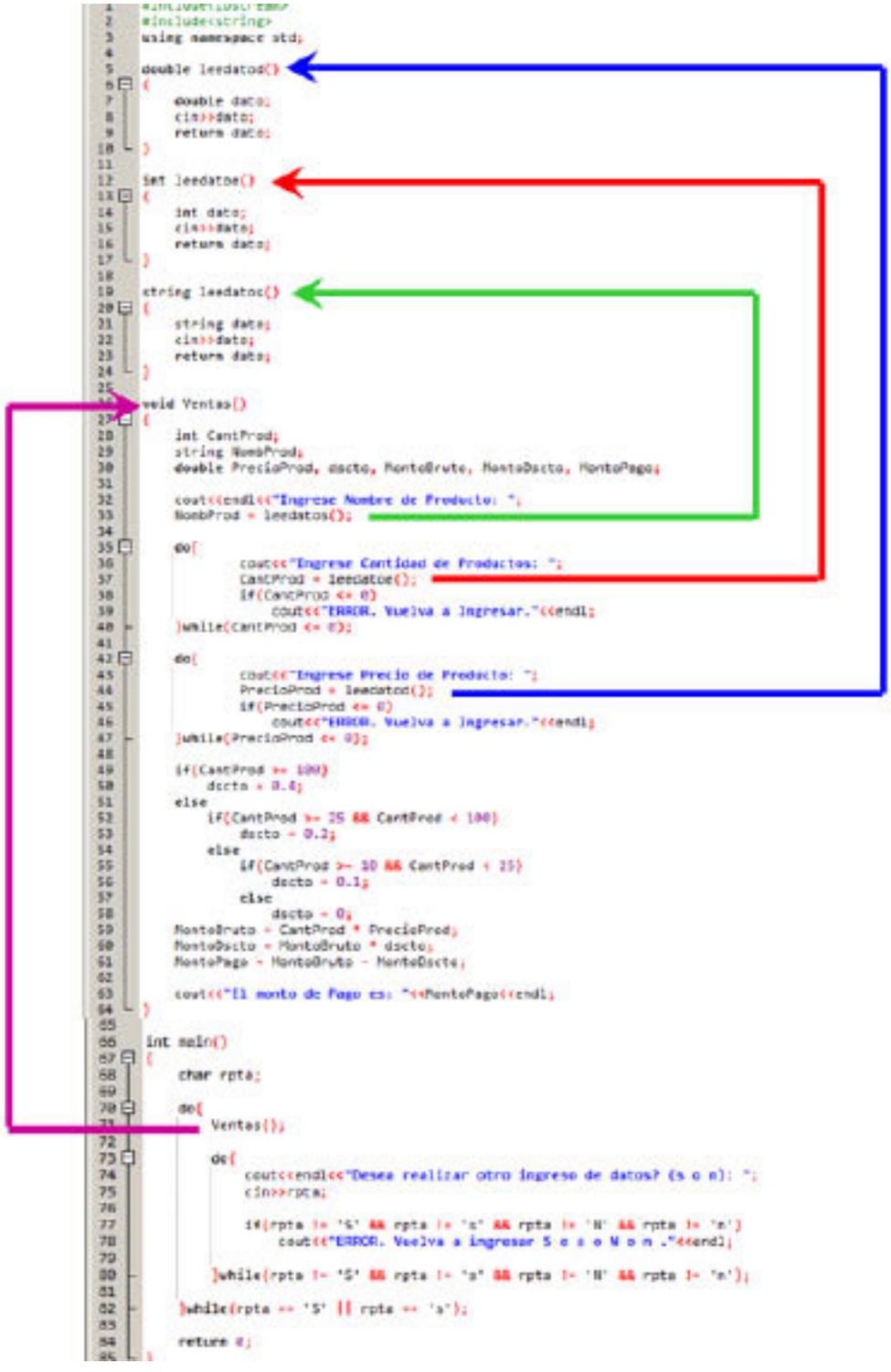
```
1  //Funciones con Argumentos
2
3  #include<iostream>
4  using namespace std;
5
6  int sumar(int a, int b);
7
8  void main()
9  {
10     int sum,a,b;
11
12     cout<<"Ingrese valor de a: \t";
13     cin>>a;
14
15     cout<<"Ingrese valor de b: \t";
16     cin>>b;
17
18     sum=sumar(a,b);
19
20     cout<<"Valor de la suma de a y b:  "<<sum;
21     cout<<"\n";
22 }
23
24 int sumar(int a, int b)
25 {
26     int c;
27
28     c=a+b;
29
30     return c;
31 }
```

Fuente: Carol Rojas M.

Otro ejemplo de aplicación de módulos de programas:

Indique el monto a pagar: Si compramos al por mayor cien o más productos nos descuentan el 40%, si compramos entre 25 y 100, el descuento es de 20% y si compramos entre 10 y 25 el descuento es de 10%. No hay descuento, si adquirimos menos de 10 productos.

Figura 87
Ejemplo1 de usar varios módulos de programa



```
#include<string>
using namespace std;

double leedatos()
{
    double dato;
    cin>>dato;
    return dato;
}

int leedatos()
{
    int dato;
    cin>>dato;
    return dato;
}

string leedatos()
{
    string dato;
    cin>>dato;
    return dato;
}

void Ventas()
{
    int CantProd;
    string NombProd;
    double PrecioProd, descto, MontoBruto, MontoDscto, MontoPago;

    cout<<endl<<"Ingrese Nombre de Producto: ";
    NombProd = leedatos();

    do{
            cout<<"Ingrese Cantidad de Productos: ";
            CantProd = leedatos();
            if(CantProd <= 0)
                cout<<"ERROR. Vuelva a Ingresar."<<endl;
    }while(CantProd <= 0);

    do{
            cout<<"Ingrese Precio de Producto: ";
            PrecioProd = leedatos();
            if(PrecioProd <= 0)
                cout<<"ERROR. Vuelva a Ingresar."<<endl;
    }while(PrecioProd <= 0);

    if(CantProd >= 100)
        descto = 0.4;
    else
        if(CantProd >= 25 && CantProd < 100)
            descto = 0.2;
        else
            if(CantProd >= 10 && CantProd < 25)
                descto = 0.1;
            else
                descto = 0;
    MontoBruto = CantProd * PrecioProd;
    MontoDscto = MontoBruto * descto;
    MontoPago = MontoBruto - MontoDscto;

    cout<<"El monto de Pago es: "<<MontoPago<<endl;
}

int main()
{
    char rpta;

    do{
        Ventas();

        do{
            cout<<endl<<"Desea realizar otro ingreso de datos? (s o n): ";
            cin>>rpta;

            if(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n')
                cout<<"ERROR. Vuelva a ingresar S o s o N o n ."<<endl;

        }while(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n');

    }while(rpta == 'S' || rpta == 's');

    return 0;
}
```

Fuente: Carol Rojas M.

Por ejemplo, si modificamos el programa de la Figura 72, usando módulos de programa, tanto para el ingreso de datos como para el cálculo, se tiene:

Figura 88
Ejemplo1 de usar varios módulos de programa

```
#include<iostream>
#include<string>
using namespace std;

char leedatoc()
{
    char dato;
    cin>>dato;
    return dato;
}

int leedatoe()
{
    int dato;
    cin>>dato;
    return dato;
}

string leedatos()
{
    string dato;
    cin>>dato;
    return dato;
}

void Calcular()
{
    string nombClie;
    char geneClie;
    double precioProd=100;
    int cantProd;
    double PorcDscto, MontoBruto, MontoDscto, MontoPago;

    cout<<"Ingrese Nombre de Cliente: ";
    nombClie = leedatos();

    do{
            cout<<"Ingrese Cantidad de Productos: ";
            cantProd = leedatoe();

            if(cantProd <= 0)
                cout<<"ERROR. Vuelva a ingresar > 0."<<endl;

    }while(cantProd <= 0);

    do{
            cout<<"Ingrese Genero de Cliente: ";
            geneClie = leedatoc();

            if(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F')
                cout<<"ERROR. Vuelva a ingresar m o M o f o F ."<<endl;

    }while(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F');
```

```
    switch(geneClie)
        {
            case 'm':
            case 'M':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.2;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.5;
                    }break;
            case 'f':
            case 'F':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.3;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.4;
                    }break;
        }

        MontoBruto = cantProd * precioProd;
        MontoDscto = MontoBruto * PorcDscto;
        MontoPago = MontoBruto - MontoDscto;

        cout<<"El monto bruto es: ";
        cout<<MontoBruto<<endl;
        cout<<"El monto descuento es: ";
        cout<<MontoDscto<<endl;
        cout<<"El monto Pago es:\t";
        cout<<MontoPago<<endl;
}

int main()
{
    char rpta;

    do{
            Calcular();
            do{
                    cout<<"Desea realizar otro ingreso de datos? (s o n): ";
                    rpta = leedatoc();

                    if(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n')
                        cout<<"ERROR. Vuelva a ingresar S o s o N o n ."<<endl;

            }while(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n');

    }while(rpta == 'S' || rpta == 's');

    return 0;
}
```

Fuente: Carol Rojas M.

LECTURA SELECCIONADA Nº 1

INTRODUCCIÓN A LOS SUBALGORITMOS O SUBPROGRAMAS

Oyanes, L. (2008). Fundamentos de Programación. Madrid: McGRAW-HILL.

Un método ya citado para solucionar un problema complejo es dividirlo en subproblemas – problemas más sencillos – y a continuación dividir estos subproblemas en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. Esta técnica de dividir el problema principal en subproblemas se suele denominar "divide y vencerás" (divide and conquer). Este método de diseñar la solución de un problema principal obteniendo las soluciones de sus subproblemas se conoce como diseño descendente (top-down design). Se denomina descendente, ya que se inicia en la parte superior con un problema general y el diseño específico de las soluciones de los subproblemas. Normalmente las partes en que se divide un programa deben poder desarrollarse independientemente entre sí.

Las soluciones de un diseño descendente pueden implementarse fácilmente en lenguajes de programación de alto nivel, como C/C++, Pascal o Fortran. Estas partes independientes se denominan subprogramas o subalgoritmos si se empelan desde el concepto algorítmico.

La correspondencia entre el diseño descendente y la solución por computadora en términos de programa principal y subprogramas se analizará a lo largo de este capítulo.

Consideremos el problema del cálculo de la superficie (área) de un rectángulo. Este problema se puede dividir en tres subproblemas:

Subproblema 1: entrada de datos de altura y base.

Subproblema 2: cálculo de la superficie.

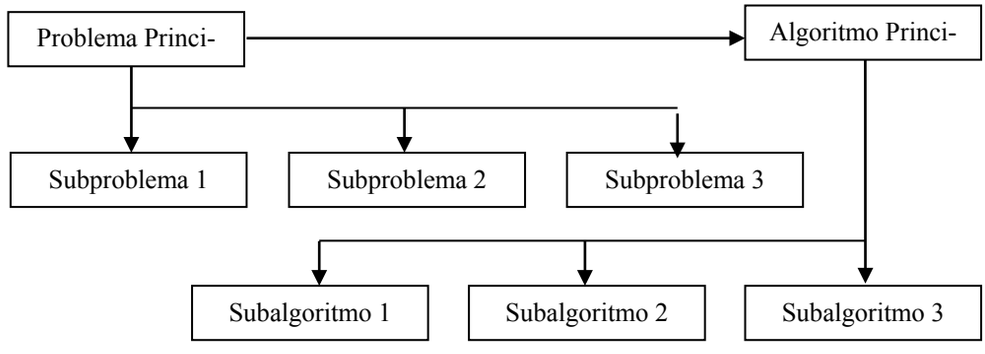
Subproblema 3: salida de resultados.

El algoritmo correspondiente que resuelve los tres subproblemas es:

```
Leer (altura, base)              //entrada de datos
Área <- base * altura            // cálculo de la superficie
Escribir (base,altura,area)      //salida de resultados
```

El método descendente se muestra en la figura:

Figura 89 Diseño descendente

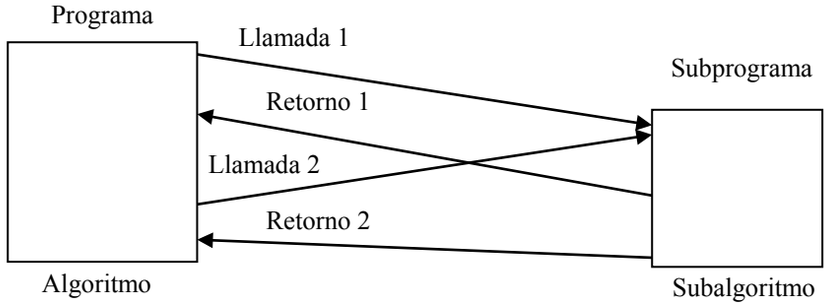


Fuente: Joyanes, L. Fundamentos de Programación

El problema principal se soluciona por el correspondiente programa o algoritmo principal – también llamado controlador o conductor (driver) – y la solución de los subproblemas mediante subprogramas, conocidos como procedimientos (subrutinas) o funciones. Los subprogramas, cuando se tratan en lenguaje algorítmico, se denominan también subalgoritmos.

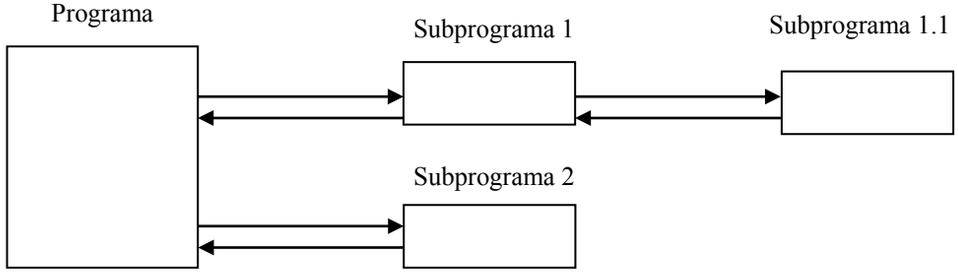
Un subprograma puede realizar las mismas acciones que un programa: 1) Aceptar datos, 2) Realizar algunos cálculos, 3) Devolver resultados. Un subprograma, sin embargo, se utiliza por el programa para un propósito específico. El subprograma recibe datos desde el programa y le devuelve resultados. Haciendo un símil con una oficina, el problema es como el jefe que da instrucciones a sus subordinados – subprogramas-; cuando la tarea termina, el subordinado devuelve un resultado al jefe. Se dice que el programa principal llama o invoca al subprograma. El subprograma ejecuta la tarea, a continuación devuelve el control al programa. Eso puede suceder en diferentes lugares del programa. Cada vez que el programa es llamado, el control retorna al lugar donde fue hecha la llamada como muestra la siguiente figura. Un subprograma puede llamar a su vez a sus propios subprogramas. Existen – como ya se ha comentado – dos tipos importantes de subprogramas: funciones y procedimientos o subrutinas.

Figura 90
Un programa con un subprograma: función y procedimiento



Fuente: Joyanes, L. Fundamentos de Programación

Figura 91
Un programa con diferentes niveles de subprogramas



Fuente: Joyanes, L. Fundamentos de Programación

ACTIVIDAD FORMATIVA N° 1

Elabora el programa de cómputo para cada situación propuesta.

INSTRUCCIONES:

1. Lee y analiza el tema N° 1 y extrae las ideas fundamentales del uso de módulos de programación.
2. Observe el vídeo **"Función y procedimiento en C++"** y complemente la información obtenida en el tema N° 1.
3. Elabore un programa en lenguaje C/C++ para las siguientes situación problema:

3.1. Realizar un programa que tenga un módulo, reciba dos argumentos a, b (enteros) y que intercambie los valores de dichas variables.

Elaborar un módulo para leer las variables a y b.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

3.2. En un cajero se pueden realizar las operaciones de depósito, saldo, retiro y salir. Elabore un programa permitir a un cliente realizar estas operaciones las veces que lo requiera (do-while).

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

3.3 Elaborar el cuadro que se muestra a continuación, sobre las funciones del C/C++, con su respectiva descripción breve y programa ejemplo definido por usted(se recomienda: http://c.conclase.net/librerias/?ansimac=isdigit).

FUNCIÓN	DESCRIPCIÓN	PROGRAMA EJEMPLO C/C++
tolower( )		
toupper( )		
isalnum( )		
isalpha( )		
isdigit( )		
isascii( )		

SINTESIS del TEMA I

Módulos de Programa

Son segmentos de código: independientes y reutilizables.

Usan Paso de Parámetros:

Por Valor
Por Referencia

Puede ser Función:
Devuelve un valor y se declara con el tipo de dato del valor que devuelve.

```
int funcionEjemplo( )
{
    return dato;
}
```

Puede ser Procedimiento:
No devuelve valor, y se declara sin dato a especificar: void.

```
void procedimientoEjemplo( )
{

}
```

TEMA N° 2: LIBRERÍAS DE PROGRAMACIÓN

¿CÓMO REUTILIZAR LOS MÓDULOS EN DIFERENTES PROGRAMAS?

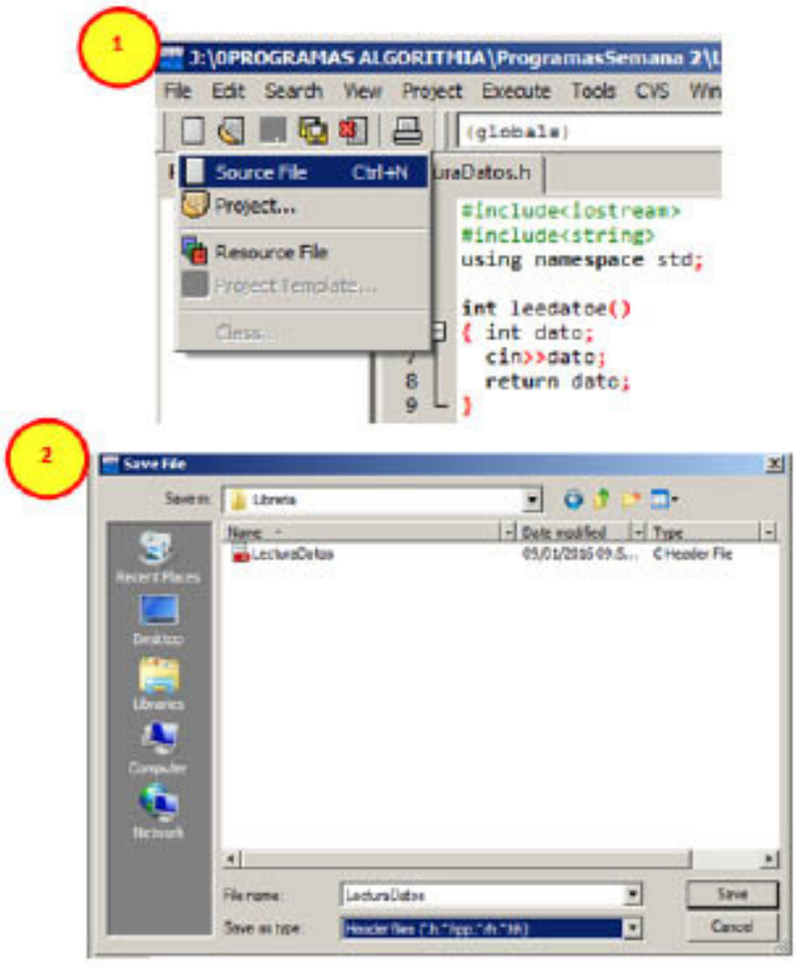
Los módulos, tanto función como procedimiento, pueden ser usados en diferentes programas sin necesidad de crearlos, una y otra vez, por lo que, en esta sección, se presenta la técnica de programación que le ayudará en lograr esta reutilización en diferentes programas.

1. DEFINICIÓN DE LIBRERÍA DE PROGRAMACIÓN

Existen librerías estándar del procesador del lenguaje C++, que se denominan de cabecera (header, por tener extensión de archivo ".h") como por ejemplo iostream.h, math.h, conio.h, stdio.h, etc. y ser invocadas en diferentes programas fuente (el programa con extensión ".cpp") como parte de la reutilización de código. Ud., como creador de programas, además de usar estas librerías, también puede crear sus propias librerías con sus propios módulos de programa.

Se puede crear la librería en cualquier editor, pero al guardar el archivo tendrá la extensión ".h"

Figura 92: Ejemplo de creación de librería en DevC++

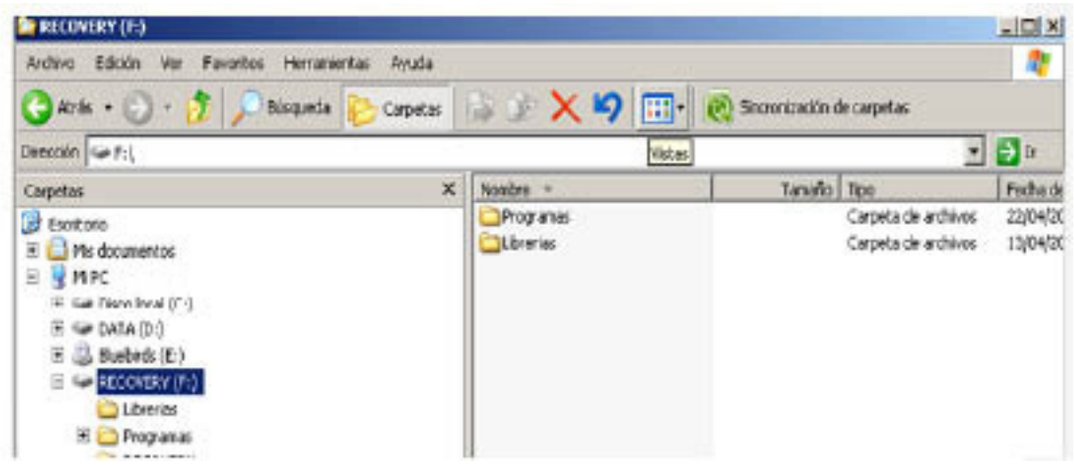


Fuente: Carol Rojas M.

Para que esta librería sea independiente a los programas, es necesario crearla en una carpeta diferente a la del proyecto o de cada programa.

De esta manera, aseguramos su reutilización entre programas, disminuyendo esfuerzo y tiempo.

Figura 93: Ejemplo de carpeta de creación de librería



Fuente: Carol Rojas M.

Elaboremos un ejemplo, creando la librería con extensión ".h" para la ingresar valores a través de función de lectura de datos de los diferentes tipos de datos, llamado "LecturaDatos.h" como se muestra en la siguiente figura.

Observe que para usar el tipo de dato "string", debe invocarse en la cabecera del programa, la librería del mismo nombre.

También considere qué algoritmo de leer datos de cada función es el mismo, pero se diferencian ya que cada módulo define un tipo de dato diferente y un nombre de módulo diferente.

Además, en la librería solo se crean los módulos y no contiene ningún módulo principal "main", ya que este existe en el programa fuente con extensión ".cpp" que lo puede compilar y ejecutar.

Figura 94: Ejemplo de librería de lectura de datos

LecturaDatos.h

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  int leedatoe()
6  { int dato;
7    cin>>dato;
8    return dato;
9  }
10
11 float leedatof()
12 { float dato;
13   cin>>dato;
14   return dato;
15 }
16
17 double leedatod()
18 { double dato;
19   cin>>dato;
20   return dato;
21 }
22
23 char leedatoc()
24 { char dato;
25   cin>>dato;
26   return dato;
27 }
28
29 string leedatos()
30 { string dato;
31   cin>>dato;
32   return dato;
33 }
```

Fuente: Carol Rojas M.

Este es un ejemplo de creación de librería para lectura de datos, sin embargo se pueden crear otras librerías con otro nombre y con otros módulos.

Pero, recuerde, que el nombre de la librería y los módulos que vaya a crear, deben estar relacionados o atender a un requerimiento, por ejemplo, puede crear una librería OperacionesBasicas.h y que contendría los módulos de sumar( ), restar( ), multiplicar( ) y dividir( ), pero de ninguna manera tendría algún modulo de leer datos.

Por ejemplo, se requiere leer un número entero diferente a cero y positivo para realizar otras acciones en el programa.

Solución:

- Crear el programa .cpp que invoca a la librería .h , "LecturaDatos.h", creada por el programador.

Figura 95: Ejemplo1 de programa que invoca a librería de lectura de datos

```
#include<iostream>
#include "LecturaDatos.h"       //Invoca a la Librería
using namespace std;

void Digitos()
{
    int num, cont=0, may, men,dig;

    do{
        cout<<"Ingrese numero: ";
        num = leedatoe();         // Invoca la módulo de la librería

        if(num < 0 )
        {
            cout<<endl<<"No se puede obtener digito. Vuelva a ingresar"<<endl.";
        }
    }while(num < 0);

    if(num == 0)
        {
            cout<<"El digito es CERO.";
        }
    else
        {   while(num>0)
            {
                dig=num%10;

                cont=cont+1;

                if(cont == 1)
                {
                    may= dig;
                    men= dig;
                }
                else
                {
                    if(dig>may)
                        may=dig;
                    else
                        if(dig<men)
                            men=dig;
                }

                num = num/10;

            }
            cout<<"El digito mayor es: "<<may<<"\n";
            cout<<"El digito menor es: "<<men<<"\n";
        }
}

int main()
{
    char rpta;

    do{
            Digitos();
            do{
                    cout<<"Desea realizar otro ingreso de datos? (s o n): ";
                    rpta = leedatoc();

                    if(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n')
                        cout<<"ERROR. Vuelva a ingresar S o s o N o n ."<<endl;

            }while(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n');

    }while(rpta == 'S' || rpta == 's');

    return 0;
}
```

Fuente: Carol Rojas M.

Otro ejemplo, y siguiendo con la modificación, esta vez de la figura 88, el programa aplica el uso de la librería LecturaDatos.h

Figura 96: Ejemplo2 de programa que invoca a librería de lectura de datos

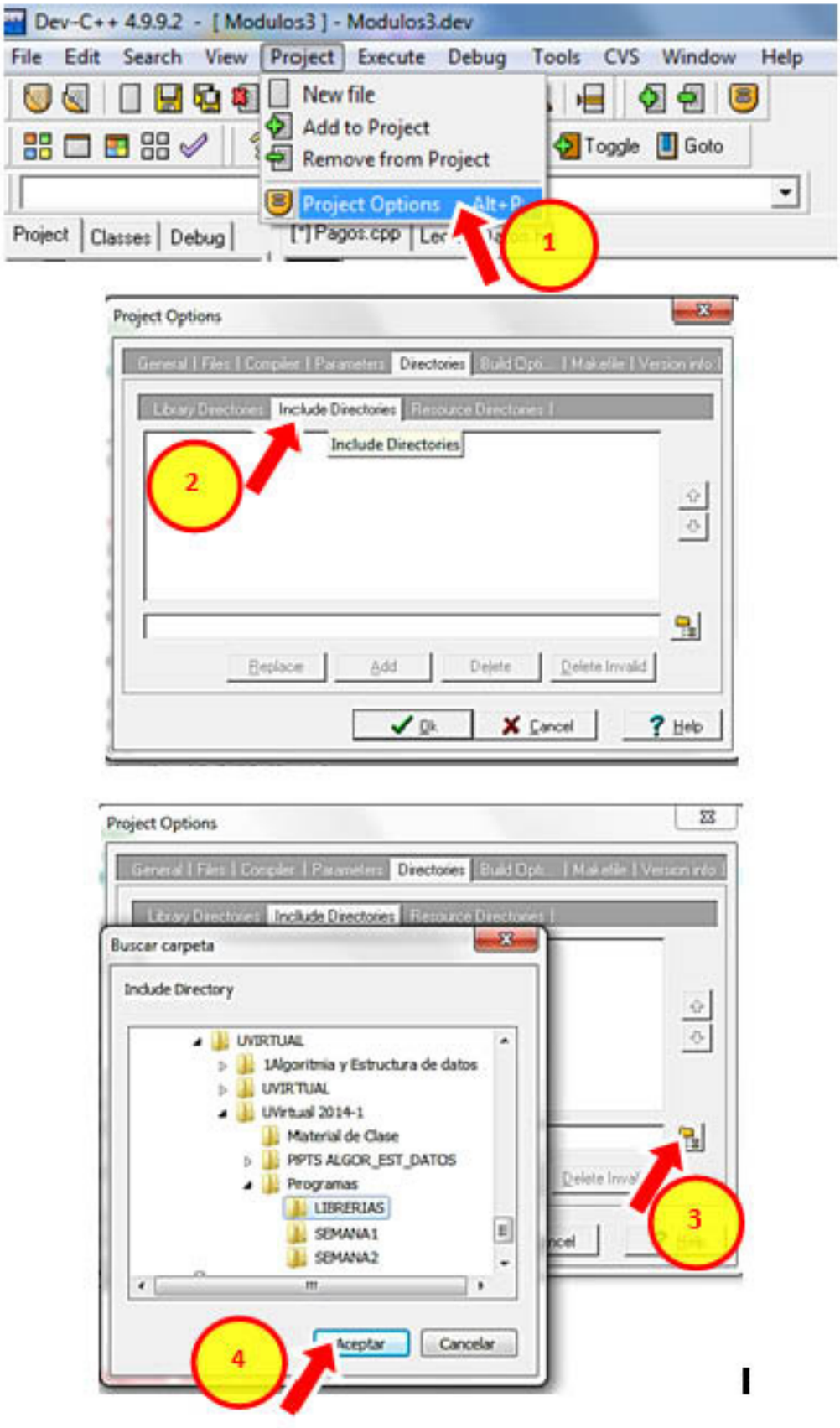
```
#include<iostream>
#include<string>
#include "LecturaDatos.h"
using namespace std;

void Calcular()
{
    string nombClie;
    char geneClie;
    double precioProd=100;
    int cantProd;
    double PorcDscto, MontoBruto, MontoDscto, MontoPago;

    cout<<"Ingrese Nombre de Cliente: ";
    nombClie = leedatos();

    do{
            cout<<"Ingrese Cantidad de Productos: ";
            cantProd = leedatoe();

            if(cantProd <= 0)
                cout<<"ERROR. Vuelva a ingresar > 0."<<endl;

    }while(cantProd <= 0);

    do{
            cout<<"Ingrese Genero de Cliente: ";
            geneClie = leedatoc();

            if(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F')
                cout<<"ERROR. Vuelva a ingresar m o M o f o F ."<<endl;

    }while(geneClie != 'm' && geneClie != 'M' && geneClie != 'f' && geneClie != 'F');

    switch(geneClie)
        {
            case 'm':
            case 'M':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.2;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.5;
                    }break;
            case 'f':
            case 'F':
                    {
                        if(cantProd <= 10)
                            PorcDscto = 0.3;
                        else
                            if(cantProd > 10)
                                PorcDscto = 0.4;
                    }break;
        }
```

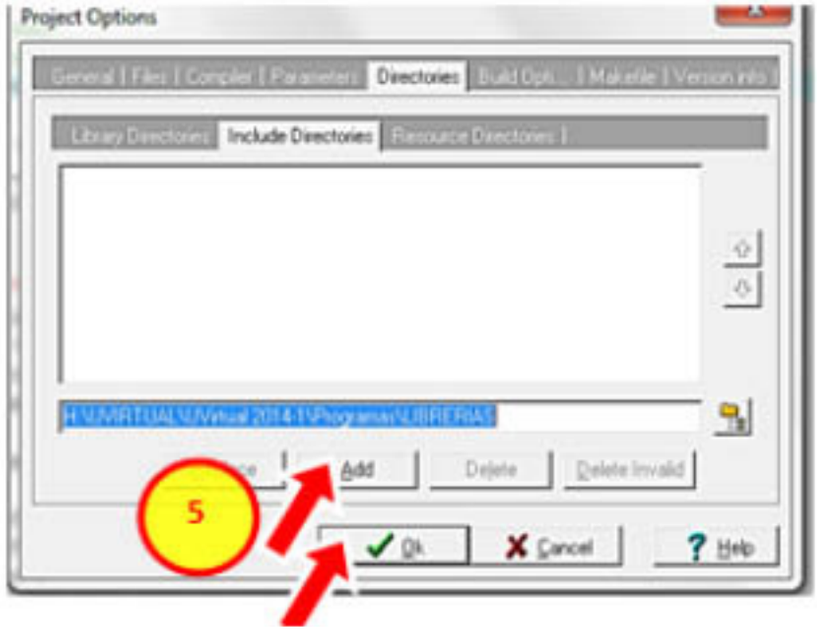
```
57          MontoBruto = cantProd * precioProd;
58          MontoDscto = MontoBruto * PorcDscto;
59          MontoPago = MontoBruto - MontoDscto;
60
61          cout<<"El monto bruto es: ";
62          cout<<MontoBruto<<endl;
63          cout<<"El monto descuento es: ";
64          cout<<MontoDscto<<endl;
65          cout<<"El monto Pago es:\t";
66          cout<<MontoPago<<endl;
67  }
68
69  int main()
70  {
71      char rpta;
72
73      do{
74              Calcular();
75              do{
76                      cout<<"Desea realizar otro ingreso de datos? (s o n): ";
77                      rpta = leedatoc();
78
79                      if(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n')
80                          cout<<"ERROR. Vuelva a ingresar S o s o N o n ."<<endl;
81
82              }while(rpta != 'S' && rpta != 's' && rpta != 'N' && rpta != 'n');
83
84      }while(rpta == 'S' || rpta == 's');
85
86      return 0;
87  }
```

Fuente: Carol Rojas M.

Se debe definir la ruta de la carpeta de la librería, antes de iniciar la compilación del programa.

Figura 97: Configurar la ruta de la librería en DevC++

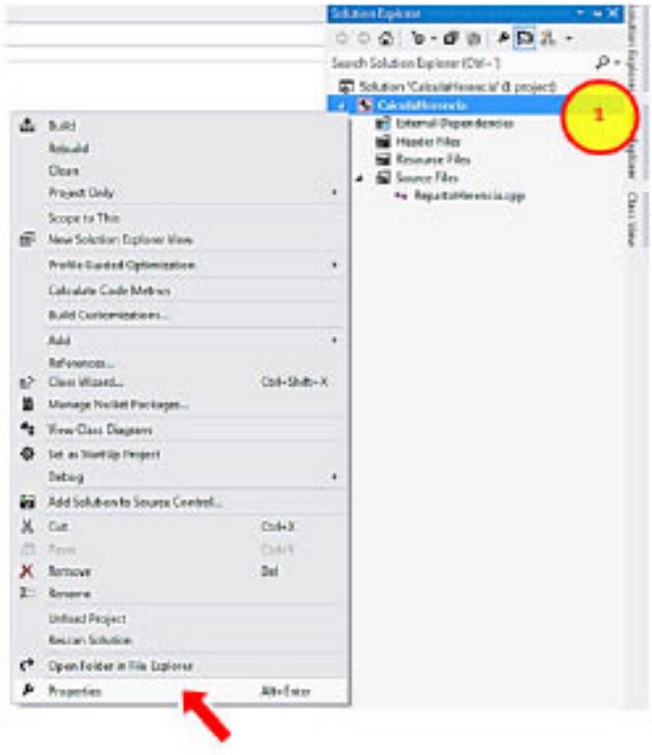


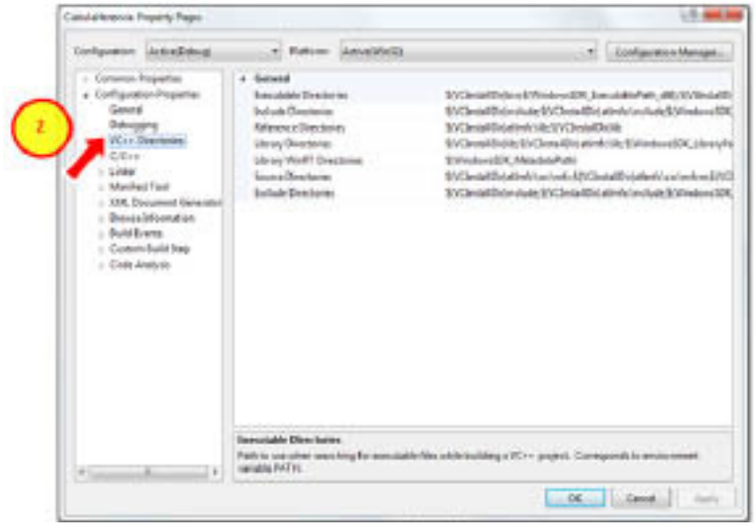


Fuente: Carol Rojas M.

En el caso de que estuviera usando otro entorno de programación, como por ejemplo el VisualStudio, debe realizar los siguientes pasos:

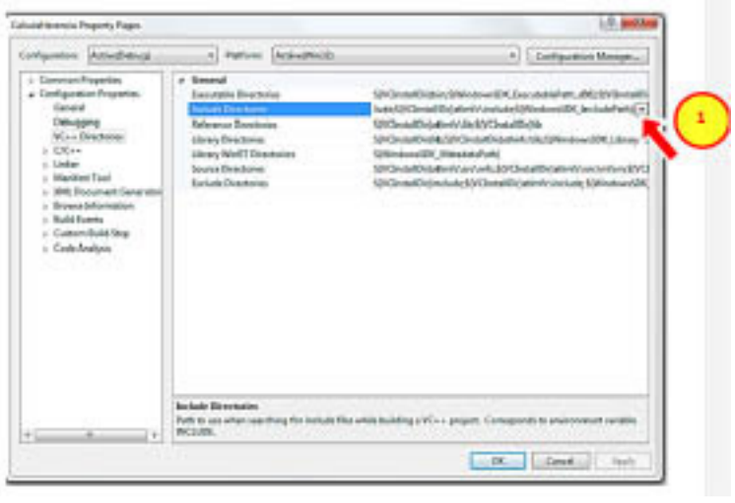
Figura 98: Configurar la ruta de la librería en VisualStudio:
Seleccionar directorios del C++ en las propiedades del proyecto

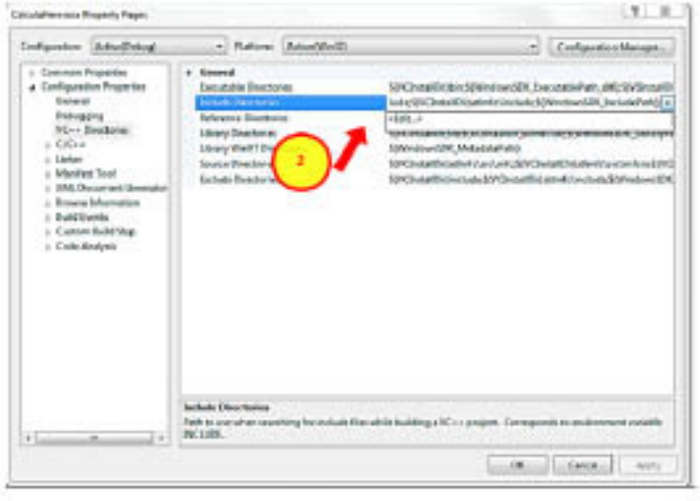




Fuente: Carol Rojas M.

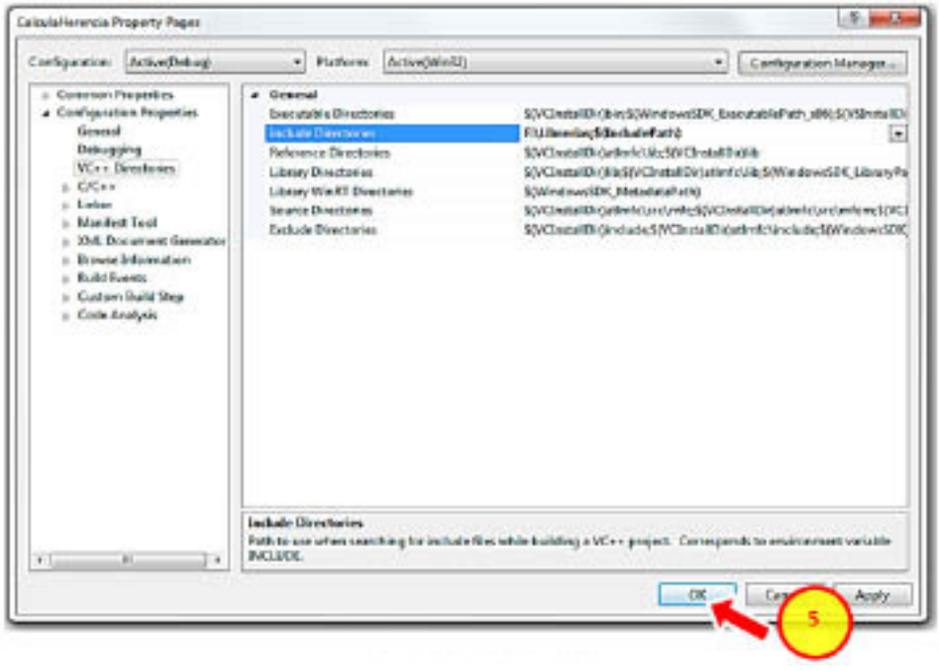
Figura 99: Configurar la ruta de la librería en VisualStudio: Modificar directorios del C++





Fuente: Carol Rojas M.
Figura

Configurar la ruta de la librería en VisualStudio:
Asignar librería a directorios del C++



Fuente: Carol Rojas M.

SINTESIS del TEMA II

Librerías de Programa

So archivos que contienen módulos independientes y reutilizables para diferentes programas.

Existen librerías propias del lenguaje, y otras que el programador puede crear.

Se invocan al inicio del programa, por eso se conoce como cabecera y con extensión .h.

```
#include <iostream>
#include "LecturaDatos.h"
```

TEMA N° 3: FUNCIONES RECURSIVAS

En cierta ocasión, en un aula de clase, pregunté si podían definir o dar un ejemplo de recursividad, sorprendiéndome la respuesta de un estudiante: "Si **yo** tengo un apuro económico, vendo **mi** reproductor de música, y atiendo ese apuro. Es decir, un recurso que le pertenece a una entidad, es usada nuevamente por la misma entidad para buscar una solución, eso es recursividad", finalizó.

1. DEFINICIÓN DE FUNCIÓN RECURSIVA

Entonces, se denominan funciones recursivas a aquellas que se invocan a sí mismas en un programa, para desarrollar un determinado proceso.

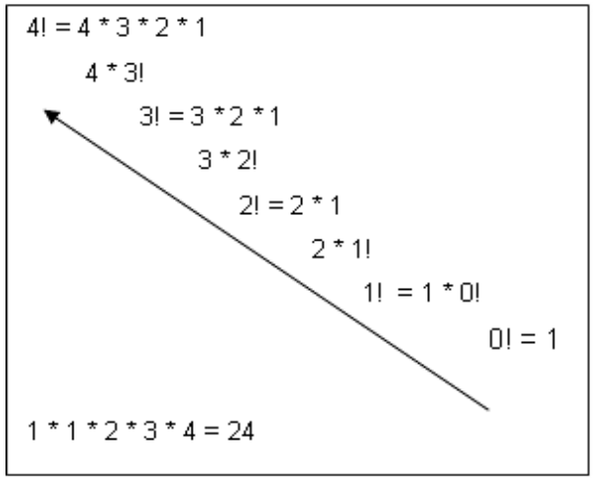
Hay que tener algunas cosas en cuenta en las funciones recursivas:

- Toda función recursiva debe tener algún punto de finalización o valor base.
- La función recursiva debe acercarse a ese punto de finalización o valor base.[13]

Por ejemplo, existe la función recursiva del factorial:

n!
4! = 4 * 3 * 2 * 1
4 * 3! → **Es decir n*(n-1)!**
3! = 3 * 2 * 1
3 * 2! → **Es decir n*(n-1)!**
2! = 2 * 1
2 * 1! → **Es decir n*(n-1)!**
1! = 1 * 0!
0! = 1

Figura 101: Ejemplo de función recursiva: Factorial de un número



Fuente: Carol Rojas M.

13 Raffo, E. (2000). Turbo C++. Lima: Mundigraph.

De las condiciones anteriores, se define el siguiente segmento de código:

```
int factorial(int n)
{
      if (n == 0)
            return 1;
      else
            return n*factorial(n-1);
}
```

Recuerde que, para los algoritmos recursivos propuestos, se debe tener en cuenta que el dato ingresado debe ser un número entero y positivo, a fin de llegar al punto de finalización al ser decrementado.

Para la recursión del factorial, se decrementa la variable "n" , a fin de que pueda llegar hasta el valor de cero, en el cual ya no se invoca a la función nuevamente, finalizando de esta manera la recursión.

Figura 102: Ejemplo de programa función recursiva: Factorial de un número

```
#include<iostream>
using namespace std;

int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n*factorial(n-1);
}

int main()
{
    int n, fact;

    do{

        cout<<"Ingrese valor de n factorial: ";
        cin>>n;

        if(n < 0)
            cout<<"ERROR. Vuelva a ingresar >= 0."<<endl;

    }while(n < 0);

    fact = factorial(n);

    cout<<"El factorial es: ";
    cout<<fact<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

Otro ejemplo de recursión, se puede dar en la multiplicación:

a * b

3 * 4 = 3 + 3 + 3 + 3

3 + (3 * 3) → **Es decir a + a*(b-1)**

3 * 3 = 3 + 3 + 3

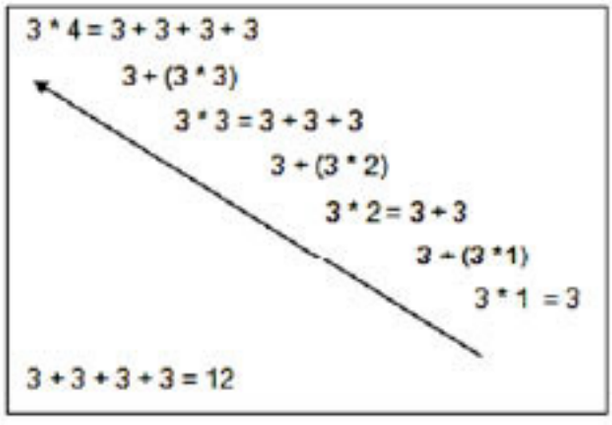
3 + (3 * 2) →**Es decir a + a*(b-1)**

3 * 2 = 3 + 3

3 + (3 *1) → **Es decir a + a*(b-1)**

3 * 1 = 3

Figura 103
Ejemplo de función recursiva: Multiplicación



Fuente: Carol Rojas M.

De las condiciones anteriores, se define el siguiente segmento de código:

```
int multiplica(int a, int b)
{
    if (a == 0 || b == 0)
       return 0;
    else
       if (b == 1)
             return a;
       else
             return a+multiplica(a,b-1);
}
```

A continuación, se muestra el programa de la función recursiva de la multiplicación, garantizando que los valores de las variables "a" y "b" sean enteros y positivos.

Figura 104
Ejemplo de programa función recursiva: Multiplicación

```
#include<iostream>
using namespace std;

int multiplica(int a, int b)
{
  if (a == 0 || b == 0)
    return 0;
  else
        if (b == 1)
            return a;
        else
            return a+multiplica(a,b-1);
}

int main()
{
    int a, b, multi;

    do{

        cout<<"Ingrese valor de variable a: ";
        cin>>a;

        if(a < 0)
            cout<<"ERROR. Vuelva a ingresar >= 0."<<endl;

    }while(a < 0);

    do{

        cout<<"Ingrese valor de variable b: ";
        cin>>b;

        if(b < 0)
            cout<<"ERROR. Vuelva a ingresar >= 0."<<endl;

    }while(b < 0);

    multi = multiplica(a,b);

    cout<<"La multiplicacion es: ";
    cout<<multi<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

Los algoritmos recursivos, no solo se pueden ser funciones, también existen procedimientos para ello, y no solo se puede invocar a sí mismo una sola vez en el mismo módulo, sino también las veces que se requiera, como los algoritmos de Torres de Hanoi, presentado a continuación.

El caso de Torres de Hanoi, el un algoritmo recursivo con procedimiento y que se invoca dos veces.

Las restricciones para la recursividad de torres de Hanoi, se expresan:

- Usar "n" discos para trasladarlos desde un poste origen, hacia un poste final, usando un poste auxiliar.

  Además "n", representa el disco más grande, y se disminuye en 1 para llegar al disco de menor tamaño.

- Por cierta cantidad de discos, hay una cantidad de movimientos, ni más ni menos, cantidad de movimientos = 2n -1.

  Si excede esta cantidad de movimientos, la ejecución del algoritmo le indicará que está erróneo.

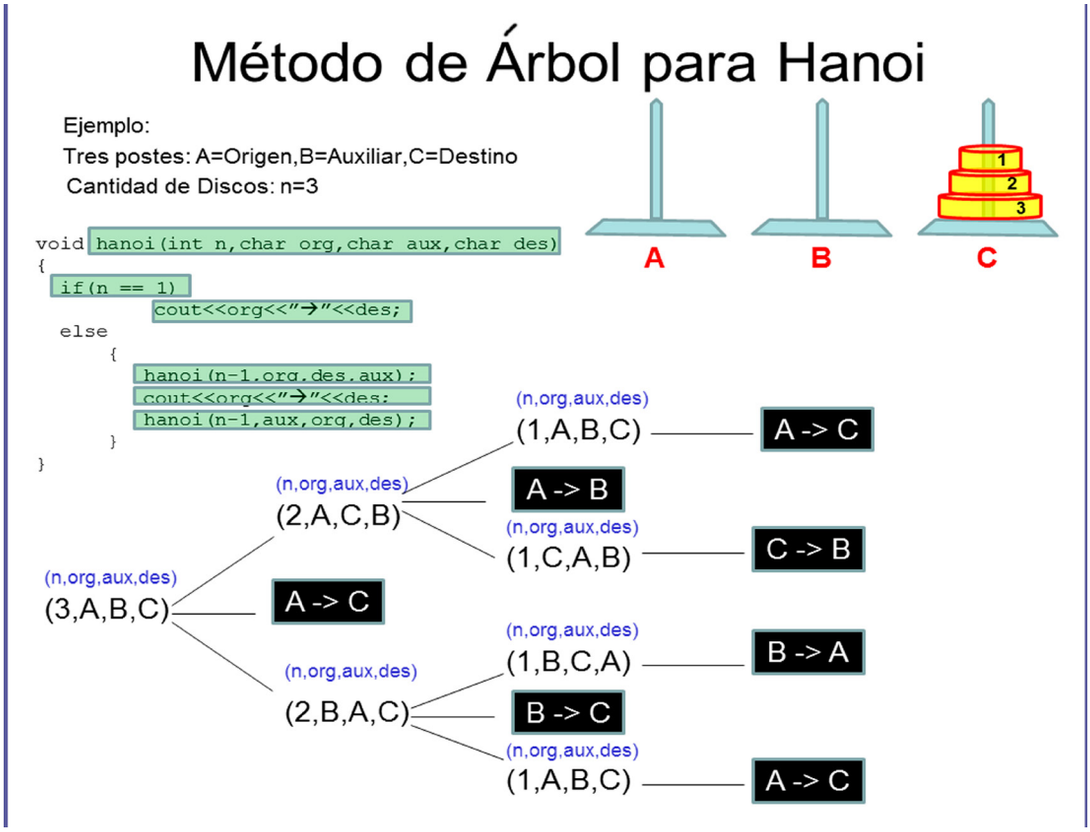
- Los discos están dispuestos del tamaño grande al más pequeño. No puede estar un disco grande sobre uno pequeño.

  Esa es la razón de la existencia de un poste auxiliar, para que pueda albergar un disco, y evitar infringir esta restricción.

  Al albergar un disco, el poste auxiliar en algún momento se convertirá en poste destino o en poste origen.

A continuación, se tiene el algoritmo puesto a prueba, a través del Método del Árbol, muy útil para comprobar la recursividad.

Figura 105
Ejemplo de procedimiento recursivo: Torres de Hanoi



Fuente: Carol Rojas M.

Figura 106
Ejemplo de programa procedimiento recursivo: Torres de Hanoi

```
#include<iostream>
using namespace std;

void hanoi(int n, char origen, char auxiliar, char destino)
{
  if(n == 1)
    cout<<origen<<" --> "<<destino<<endl;
  else
      {
         hanoi(n-1, origen, destino, auxiliar);
         cout<<origen<<" --> "<<destino<<endl;
         hanoi(n-1, auxiliar, origen, destino);
      }
}


int main()
{
    int n;

    do{

        cout<<"Ingrese cantidad de discos: ";
        cin>>n;

        if(n <= 0)
            cout<<"ERROR. Vuelva a ingresar >= 0."<<endl;

    }while(n <= 0);

    cout<<endl;

    // Invocar a Hanoi, enviando discos y los postes A, B, C
    hanoi(n, 'A', 'B', 'C');

    cout<<endl;

    return 0;
}
```

Fuente: Carol Rojas M.

También se puede crear una librería con estos módulos recursivos y poder reutilizarlos en otros programas, con mayor facilidad: FRecursivas.h.

Figura 107
Ejemplo de librería con módulos recursivos

```
#include<iostream>
using namespace std;

int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n*factorial(n-1);
}

int multiplica(int a, int b)
{
  if (a == 0 || b == 0)
    return 0;
  else
        if (b == 1)
            return a;
        else
            return a+multiplica(a,b-1);
}

void hanoi(int n, char origen, char auxiliar, char destino)
{
  if(n == 1)
    cout<<origen<<" --> "<<destino<<endl;
  else
      {
          hanoi(n-1, origen, destino, auxiliar);
          cout<<origen<<" --> "<<destino<<endl;
          hanoi(n-1, auxiliar, origen, destino);
      }
}
```

Fuente: Carol Rojas M.

Existen otros algoritmos recursivos, que usted Puede investigar y aplicar en sus programas, como por ejemplo la recursión de la potencia, de la serie Fibonacci, del máximo común divisor entre otras.

SINTESIS del TEMA III

Funciones Recursivas

Generan un valor, y vuelven a usarlo sin terminar la ejecución del módulo.

También existen procedimientos recursivos.

Existen algoritmos recursivos definidos, y otros que puede crear el programador.

Ejemplos de Algoritmos Recursivos:

- Potencia.
- Fibonacci
- Sumar
- Mostrar Lista
- Mostrar Grafo

ACTIVIDAD FORMATIVA N° 2

Elabora el programa de cómputo para cada situación propuesta.

INSTRUCCIONES:

1. Lee y analiza los temas N° 2 y N° 3 y extrae las ideas fundamentales del uso de las librerías de programación y de los algoritmos recursivos.
2. Observe el vídeo "Tutorial 15 de C++ - Headers (Como crear librerías)" y complemente la información obtenida en el tema N° 2.
3. Observe el vídeo "Función Fibonacci recursiva. Ejemplo" y complemente la información obtenida en el tema N° 3.
4. Elabore un programa en lenguaje C/C++ para las siguiente situación problema:

4.1 Para repartir una herencia se tiene en cuenta: Si la cantidad de hijos es menor a cuatro, se repartirá exactamente entre el número de hijos; si son cuatro o más hijos, la mitad le tocará al hermano mayor y el resto se dividirá entre los demás hermanos. Valide los datos ingresados, solicite si desea seguir con varios ingresos de repartición y use la librería de Lectura de datos.

IDENTIFICAR VARIABLES A USAR	VARIABLE(S) A INGRESAR	PROCESO (CÁLCULO)	VARIABLE(S) PARA REPORTAR

4.2 Elaborar los módulos recursivos, considerando una prueba de escritorio, realizado en Word, para:

a. Potencia

b. Serie Fibonacci

El instrumento para calificar es una lista de cotejo con los siguientes criterios, para un archivo Word con la tabla de las partes del algoritmo, y el archivo fuente (.cpp) y el archivo de cabecera (.h) de cada ejercicio, en una carpeta ApellidoNombreAlumno.zip:

	CRITERIO	PUNTAJE
Ejercicio 4.1	Elaboración de la tabla con las tres partes del algoritmo y las variables a usar.	2
	Crear el módulo (función o procedimiento, según se requiera) que calcula la herencia, y que será invocada por el módulo principal.	2
	Declarar las variables con su respectivo tipo de dato.	2
	Ingresar cada dato, usando los módulos de la librería LecturaDatos.h.	2
	Validar cada dato ingresado, usando la sentencia repetitiva hacer-mientras, por cada dato.	2
	Calcular la herencia según las condiciones dadas en el enunciado.	2
	Mostrar el valor repartido de la herencia, según cada condición.	2
	Crear el módulo principal, que invoca al módulo que calcula, para una cierta cantidad de veces, sentencia repetitiva hacer-mientras.	2
Ejercicio 4.1	Modulo recursivo de la potencia y de la serie Fibonacci.	2
	Prueba de escritorio (a mano alzada en Word) de cada módulo recursivo	2

GLOSARIO DE LA UNIDAD IV

A

ARGUMENTO.
Valores suministrados a un módulo de programa, al momento de ser invocado.

C

CABECERA.
Sección inicial del programa donde se declara el uso de librerías para que el programa logre ser compilado (traducido y ejecutado).

COMPILACIÓN.
Es la acción e compilar, es decir, traducir del programa fuente al código máquina.

I

INVOCACIÓN.
Es la acción de llamar a ejecución a un módulo, en otro módulo, en otra librería o en el módulo principal.

L

LIBRERÍA.
Es un archivo, que contiene código, usualmente en módulos de programa. Que pueden ser reutilizados por diferentes proyectos de programa.

O

OPERADOR DE DIRECCIÓN.
Permite a una variable, almacenar valores en hexadecimal específicamente direcciones de memoria del computador.

S

STRING.H.
Librería del compilador, en este caso el C/C++, que permite usar el tipo de dato del mismo nombre: string.

V

VARIABLE GLOBAL.
Espacio de memoria, que se declara en la cabecera del programa, y puede usarse en todos los módulos del mismo programa.

BIBLIOGRAFIA DE LA UNIDAD IV

- **Joyanes, L. (2008).** *Fundamentos de programación*. Madrid: McGRAW-HILL.
- **Raffo, E. (2000).** *Turbo C++*. Lima: Mundigraph.

AUTOEVALUACION Nº 4

1. El módulo de programa, el paso de parámetros por valor, realiza el siguiente proceso:

   a) Crea una variable original y las modificaciones afectan a la copia de esta.

   b) Crea un variable original y las modificaciones solo afectan a ésta.

   c) Crea una copia de la variable y las modificaciones afectan a la copia.

   d) Crea una copia de la variable y las modificaciones afectan a la original.

   e) Crea una variable original y no tiene modificaciones.

2. Indique la alternativa que indique la(s) variables(s) que sean paso de parámetros por referencia.

```
void calcular(int x, int  &y)
{      x++; y ++;      }
```

   a) La variable "x"

   b) La variable "x" y la variable "y"

   c) La variable "y"

   d) La variable "c" y la variable "y"

   e) La variable "c"

3. Indique la alternativa correcta para definir una función.

   a) Es módulo de programa y no devuelve valor.

   b) Es módulo de programa y nunca usa argumentos.

   c) Es módulo de programa y devuelve solo varios valores.

   d) Es módulo de programa y debe argumentos obligatoriamente.

   e) Es módulo de programa y devuelve solo un valor.

4. Indique la alternativa correcta para definir un procedimiento.

   a) Es módulo de programa y debe argumentos obligatoriamente.

   b) Es módulo de programa y no devuelve valor.

   c) Es módulo de programa y devuelve solo varios valores.

   d) Es módulo de programa y nunca usa argumentos.

   e) Es módulo de programa y devuelve solo un valor.

5. Indique la alternativa cuyo enunciado no es correcto:

   a) Las funciones y procedimientos son módulos de programa.

   b) Las funciones recursivas se invocan a sí mismas.

   c) Sólo las funciones usan argumentos, los procedimientos no lo usan.

   d) Los módulos son independientes y reutilizables.

   e) Los pasos de parámetros son dos: por valor y por referencia.

6. Indique la alternativa que defina a "permite usar módulos independientes, en diferentes programas".

   a) Función recursiva

   b) Paso de parámetros por valor

   c) Sentencias repetitivas

   d) Librerías de programación

   e) Sentencias selectivas

7. Respecto a la librería de programación, no es una característica que le corresponda:

   a) Tiene módulo principal: int main( ).

   b) Se guardan como un archivo de cabecera ".h".

   c) Tiene módulos que el programador crea.

   d) Es invocado en diferentes programas.

   e) Los módulos creados es para un mismo requerimiento.

8. Indique a qué algoritmo recursivo se refiere el siguiente módulo de programa:

```
int módulo(int a, int b)
{
  if (a == 0 || b == 0)
        return 0;
  else
if (b == 1)
return a;
else
return a + modulo(a,b-1);
}
```

a) Función recursiva de la potencia

b) Función recursiva de la multiplicación

c) Función recursiva de torres de Hanoi

d) Función recursiva del factorial

e) Función recursiva de Fibonacci

9. Indique a que algoritmo recursivo, se refiere el siguiente módulo de programa:

```
int modulo(int a, int b)
{
  if(b==0)
      return 1;
  else
      return a * modulo(a,b-1);
}
```

a) Función recursiva de la multiplicación

b) Función recursiva del factorial

c) Función recursiva de torres de Hanoi

d) Función recursiva de la potencia

e) Función recursiva de Fibonacci

10. En el algoritmo recursivo de las torres de Hanoi, se tiene la variable "n", indique el significado en el algoritmo.

a) Es la cantidad de movimientos

b) Es la cantidad de discos

c) Es la cantidad de veces de la recursión

d) Es la cantidad de postes

e) Es la cantidad de postes auxiliares

ANEXO Nº 1

RESPUESTAS DE LA AUTOEVALUACIÓN DE LA UNIDAD I

N°	RPTA.
1	B
2	D
3	E
4	C
5	C
6	E
7	C
8	A
9	A
10	D

RESPUESTAS DE LA AUTOEVALUACIÓN DE LA UNIDAD II

N°	RPTA.
1	E
2	B
3	D
4	A
5	E
6	C
7	E
8	A
9	E
10	D

RESPUESTAS DE LA AUTOEVALUACIÓN DE LA UNIDAD III

N°	RPTA.
1	E
2	A
3	C
4	D
5	C
6	B
7	B
8	D
9	C
10	E

RESPUESTAS DE LA AUTOEVALUACIÓN DE LA UNIDAD IV

N°	RPTA.
1	C
2	C
3	E
4	B
5	C
6	D
7	A
8	B
9	D
10	B

UNIVERSIDAD CONTINENTAL | MODALIDAD VIRTUAL

Este manual autoformativo es el material didáctico más importante de la presente Asignatura, desarrollada para la modalidad virtual. Elaborado por el docente, orienta y facilita el autoaprendizaje de los contenidos y el desarrollo de las actividades propuestas en el sílabo.

Los demás recursos educativos del aula virtual complementan y se derivan del manual. Los contenidos multimedia ofrecidos utilizando videos, presentaciones, audios, clases interactivas, se corresponden a los contenidos del presente manual.

La modalidad te permite estudiar desde el lugar donde se encuentres y a la hora que más le convenga. Basta conectarte a la Internet, ingresar al campus virtual donde encontrarás todos tus servicios: aulas, videoclases, presentaciones animadas, biblioteca de recursos, muro y las tareas, siempre acompañado de tus docentes y amigos.

El modelo educativo de la universidad continental virtual es innovador, interactivo e integral, conjugando el conocimiento, la investigación y la innovación. Su estructura, organización y funcionamiento están de acuerdo a los estándares internacionales. Es innovador, porque desarrolla las mejores prácticas del e-learning universitario global; interactivo, porque proporciona recursos para la comunicación y colaboración síncrona y asíncrona con docentes y estudiantes; e integral, pues articula contenidos, medios y recursos para el aprendizaje permanente y en espacios Flexibles.

MANUALES **AUTOFORMATIVOS**