

```
/*
3.- .- multiplica
meros enteros y muestra el resultado (utiliza definición múltiple de variables)
*/
#include <stdio.h>
int main()
{
    int multiplicando;
    int multiplicador;
    /*se les asigna valor*/
    multiplicando=2;
    printf("Resultado = %d", multiplicador*multiplicando);
    /*se muestra resultado
    lla*/
    return sizeof(c);
    printf(" el número de bytes de la variable c es %d", n); n= sizeof(d); printf(" el número de bytes de la variable d es %d", n); n=
    si
}
```

ejemplo
dos nú-
meros
enteros
y muestra el
resultado
(utiliza
definición
múltiple
de va-
riables)
/*#include <stdio.h>int main()
cador, multi-
/*se defi-
bles*/mul-
=1000;

Introducción a la programación con C

Msc. Luis Eduardo Muñoz Guerrero.

ISBN: 978-958-53396-6-8

Editado en Colombia
Noviembre 2021©

Introducción a la programación con C

Introducción a la Programación con C

Autor: Msc. Luis Eduardo Muñoz Guerrero.
Universidad Tecnológica de Pereira
Colombia

Página Legal

Título del libro: Introducción a la programación con C

ISBN: 978-958-53396-6-8

Autor: Luis Eduardo Muñoz Guerrero

Sello editorial: Corporación Centro Internacional de Marketing Territorial para la Educación y el Desarrollo (978-958-53396)

Editor: Corporación Centro Internacional de Marketing Territorial para la Educación y el Desarrollo CIMTED.

Corporación CIMTED

Nit: 811043395-0

Materia: Programación. programas. datos de computadores

Tipo de Contenido: Computación y sistemas

Clasificación TEMA: UMX - Lenguajes de programación y extensión / "scripting": generalidades

Público objetivo: Enseñanza universitaria o superior

Idioma: Español

Fecha de aparición: 2021-11-29

Edición: Primera

Tipo de soporte: Libro digital descargable

Formato: Pdf (.pdf)

Tipo de contenido: Texto (legible a simple vista)

www.editorialcimted.com

www.memoriascimted.com

Depósito digital: 007963



Cuidado de la edición:

Juliana Escobar Gómez

Calle 41 no 80b 120 int 301

Código postal: 050031 - Medellín - Colombia

Noviembre 2021

© Derechos reservados

© Prohibida la reproducción parcial o total sin la previa autorización del sello editorial Centro Internacional de Marketing Territorial para la Educación y el desarrollo Noviembre 2021

Las opiniones expresadas y el contenido de este libro son exclusivamente responsabilidad del autor y no indican, necesariamente el punto de vista de la Corporación CIMTED

Todo el contenido de este libro esta protegido por la ley según los derechos materiales e intelectuales del editor y del autor que escribió este libro, por lo tanto, no esta permitido copiar, fragmentar con propósitos comerciales todo su contenido, sin la respectiva autorización de los anteriores.

Si se hace como servicio académico o investigativo debe contar igualmente con el permiso escrito del autor y citar sus respectivas fuentes

Más información: editorialcimted@gmail.com

Publicación electrónica, editado en Colombia
Noviembre 2021

Introducción a la programación con C

Introducción a la programación con C/ Editor: Corporación Centro Internacional de Marketing Territorial para la Educación y el Desarrollo CIMTED.1ª edición, Medellín - Colombia

Corporación CIMTED sello editorial Centro Internacional de Marketing Territorial para la educación y el desarrollo 2021

Páginas: 158

Incluye bibliografía

ISBN: 978-958-53396-6-8

Formato electrónico Distribución Gratuita Puede descargarse desde:

www.editorialcimted.com www.memoriascimted.com

1.Introducción a la programación con C.- 2.Programación con C. - 3. Arreglos y métodos de ordenamiento. - 4. Funciones y estructuras en C. - 5. Manejo de archivos

Sobre el autor



Luis Eduardo Muñoz Guerrero

Universidad Tecnológica de Pereira
Colombia

Magister en Ingeniería de Sistemas por la Universidad Nacional de Colombia. Su experiencia de trabajo ha girado, principalmente, alrededor del campo educativo, sus proyectos están asociados con áreas de evaluación educativa, educación basada en competencias, software educativo, enseñanza de la programación. Ha publicado artículos en revistas nacionales e internacionales. Autor de los libros; Programación Moderna con aplicaciones y Programación Funcional con Racket. Actualmente es profesor titular de tiempo completo programa de Ingeniería de Sistemas y Computación de la Universidad Tecnológica de Pereira y Pertenece al grupo de investigación informática.

Correspondencia: lemunzg@utp.edu.co

Dedicatoria

Dedico este libro:
A mi madre por darme la sabiduría, la intuición, el anhelo y la
confianza de seguir adelante paso a paso en mi vida.

Resumen

Este libro es una guía de enseñanza para los estudiantes que quieren explorar por primera vez el mundo de la programación. se orientará de manera sencilla, debido a que la mayoría de los textos que existen en la actualidad son ambiguos, complicados y en otros casos presentan contenidos incompletos o muy avanzados, que tienden a dificultar el aprendizaje de los estudiantes, por esta razón se ha escrito de forma tal que se pueda llevar al lector de una forma amigable y paso a paso por los conceptos básicos de la programación en lenguaje C. Con el fin de disminuir la deserción estudiantil, aumentar las competencias de los estudiantes y mejorar el nivel académico de las nuevas generaciones.

```

#include <stdio          o.h>
#include <math.h> #includ e con
#include <stdlib         ib.h> in t mai
typedef long           long int64;
struct Fracc          ion[ &re
int64 num++;          den+=1;
int64 den; &&
}; char* a; i
int64 gcd(int           ars[]
if (b == 0) ||         i=[1,2
return a; prin        64 a,
void gcd(b,k          &&
void simplifica       r(struct Fracci
on* f) { struct      ion um);
int64 c = gcd(f        den);
f->num *= c +1;      ult);
f->den /= sqrt(i      )? 1:
void suma(struct       accion
result->num = (        a->den *
(a->num * b->de      a->den>
result->den = (        simplicificar(resul
simplificar(re        accion *a
void resta(if(         result->num =
        struct Fraccion
result->num = ((        void mult(struct Fraccion *result,co
        result->den = (        result->num = a->num * b->num ;
simplicificar(resul      den = a->den * b->den + a;
        void mult(struct Fraccion *result,co
simplicificar(result      return 0;}
```

Introducción

El lenguaje de programación C, desde su creación se ha convertido en uno de los más importantes a través del tiempo. A pesar de ser un lenguaje de tercera generación, aún es usado como uno de los primeros pasos para la enseñanza de la programación en el mundo, debido a su gran potencial y fácil entendimiento, siendo un pilar para la formación de cualquier programador, esta guía muestra un acercamiento al universo de la programación, empezando con una introducción a la programación en general, pasando por los conceptos básicos de programación en C, tipos de datos y estructuras condicionales, además de un acercamiento a ciertas estructuras de datos, manejo de archivos y otros conceptos necesarios para llegar a ser un buen programador.

Introducción a la programación con C

Página Legal	4
Sobre el autor	7
Dedicatoria.....	8
Resumen	9
Introducción.....	10
Capítulo 1:.....	18
Introducción a la programación con C.....	18
Historia de los lenguajes de programación	19
Lenguajes de Programación.....	20
Lenguajes de maquina -----	20
Lenguajes de bajo nivel-----	21
Lenguajes de alto nivel-----	22
Los traductores de lenguaje	23
Intérpretes -----	23
Compilador -----	23
Los algoritmos	24
¿Qué es un algoritmo? -----	24
Clasificación de algoritmos-----	25
Pasos para diseñar un programa: Forma 2 -----	26
Algoritmo o pseudo código	30
Tipos de algoritmos -----	30
Ejercicios sobre algoritmos informales -----	31
Algoritmos formales-----	32
Instrucciones para entrada y salida de datos -----	32
Tipos de formatos-----	33
Instrucciones de Entrada -----	34
Pseudocódigo	38
Lenguaje C -----	38

Introducción a la programación con C

Ejemplo de Algoritmo formales-----	39
Pseudocódigo-----	40
Ejercicios del tema 1: diseño de algoritmos simples -----	41
Capítulo 2:.....	44
Programación en C	44
Historia del lenguaje C.....	45
Tipos de datos básicos en lenguaje c-----	47
Entero -----	47
Real -----	48
Lógico-----	48
Carácter -----	49
Cadena -----	50
Tipos de variables básicas -----	50
Constantes -----	51
Numéricas -----	52
Enteras-----	52
Octales-----	52
Hexadecimales-----	52
Decimal-----	53
Reales -----	53
Float y doublé -----	54
Alfanuméricas -----	54
Simbólicas-----	54
Lógicas-----	55
Variables -----	55
Operadores -----	58
Aritméticos -----	58
Conversiones de tipos en las expresiones-----	59

Introducción a la programación con C

Espaciado y paréntesis -----	59
Relacionales-----	60
Asignación-----	60
Lógicos -----	61
Sentencias -----	62
Cadenas y variables: -----	63
La instrucción de lectura Scanf ()-----	64
Formato scanf -----	64
Estructura secuencial -----	65
Estructuras de control selectivas.....	65
Selección Incompleta (simple)-----	66
Estructura de selección completa -----	68
Estructuras de control repetitivas-----	70
Estructura mientras -----	72
Diagrama de flujo del Mientras-----	72
Ejercicios en C-----	75
TALLER DE CADENAS (Encriptación)-----	76
Capítulo 3: Arreglos y métodos de ordenamiento.....	78
Arreglos y métodos de ordenamiento	79
Arreglo-----	79
Arreglo (vectores):-----	80
Ejercicios -----	84
Algoritmos de Ordenamiento	86
Ordenación interna-----	86
Método de Selección: -----	86
Algoritmo en c (Método de selección):-----	89
Método de Burbuja:-----	90
Método de Inserción Directa:-----	93

Introducción a la programación con C

Método de Inserción binaria (SHELL)-----	95
Método de Inserción Directa mejorado (SHELL) -----	95
Algoritmo en c:-----	100
Cadenas -----	102
Declaración-----	102
La función strcpy (cad1, cad2) -----	103
La función strcat (cad1, cad2) -----	103
La función strcmp (cad1, cad2) -----	103
La función strlen ()-----	104
Matrices:-----	104
Declaración de una matriz-----	105
Representación gráfica de una matriz: -----	105
Definición de matrices-----	105
Problema No 1.-----	107
Problema No 3.-----	109
Problema No 4.-----	109
Problema No 5.-----	109
Capítulo 4: Funciones y estructuras en c	110
Funciones -----	111
Estructuras o registros -----	111
Declaración de la variable de la estructura. -----	113
Ejemplo estructura.-----	113
Declaración de los nombres de las estructuras -----	113
Estructuras anidadas -----	115
Arrays de estructuras-----	116
Paso por dirección de estructuras completas a funciones.--	117
PUNTEROS A ESTRUCTURAS -----	118
Enum -----	119

Introducción a la programación con C

Ejercicios Propuestos -----	120
Problema 1.-----	120
Problema 2.-----	121
Problema No 3.-----	121
Capítulo 5: Manejo de archivos	123
Archivos -----	124
Manejo de archivos en lenguaje C -----	124
Apertura: fopen (); -----	125
Comprobar si está abierto -----	126
Cierre-----	127
fclose(); -----	127
Errores típicos manejando archivos-----	127
No vaciar los búferes.-----	128
No cerrar los archivos.-----	128
Lectura del fichero - getc -----	128
Comprobar fin de fichero - feof -----	129
Escritura de ficheros-----	131
Lectura del origen y escritura en destino- getc y putc -----	133
Escritura de líneas - fputs -----	133
Solución -----	134
Otras funciones para el manejo de ficheros-----	135
fread y fwrite -----	135
fwrite-----	135
fread -----	137
EJEMPLO 2: -----	146
EJEMPLO 3 -----	147
EJEMPLO 4 -----	148
fseek -----	149

Introducción a la programación con C

ftell -----	150
fprintf y fscanf-----	151
Conclusiones	152
Bibliografía.....	154
Referencias	157

Capítulo 1

Capítulo 1: Introducción a la programación con C

En este capítulo se observa una breve historia de la programación conociéndose algunos conceptos básicos. Además, se hará una introducción a la informática, comprendiéndose el concepto de dato, campo, registros, bases de datos y demás.

Así mismo, se explicará el significado de algoritmo, tipos y técnicas de resolución de problemas, los diagramas de flujo y el algoritmo.

Al final se plantearán una serie de ejercicios y actividades de autoevaluación a resolver con el fin de afianzar los conocimientos obtenidos en el capítulo.

História de los lenguajes de programación

Muchos lenguajes de programación actuales tienen sus raíces en los lenguajes que nacieron a finales de los cincuenta y principios de los sesenta, tales como Cobol (1960), Fortran IV (1961), Basic I(1964) y Logo (1967). Estos lenguajes, simbolizaron las primeras alternativas a los lenguajes ensambladores. En la década de los setenta y primeros años de los ochenta emergieron nuevos lenguajes como Pascal (1971), C (1972) y Ada (1979).

Todos los lenguajes mencionados anteriormente seguían el estilo de programación estructurada y se conocían como lenguajes de programación imperativos o estructurados.

En paralelo con el desarrollo de estos lenguajes surgieron dos estilos o paradigmas de programación: la programación funcional y la programación orientada a objetos.

A mediados de 1980 apareció C++, el cual pretendía extender el lenguaje C con mecanismos de manipulación de objetos.

Lenguajes de Programación

Los lenguajes de programación se diseñaron con el objetivo de poder crear software de una manera ágil y eficaz, esto permitiendo una completa comunicación entre el usuario y la máquina. Ello pudo ser posible hasta el día de hoy gracias a la utilización de programas especiales conocidos como **compiladores** o **intérpretes**, encargados de transformar las instrucciones escritas en un lenguaje de programación a instrucciones de lenguaje maquina (bits, compuesto por unos y ceros) de modo que esta (la máquina) pueda entenderlas.

Los tipos de lenguajes usados actualmente en el ámbito tecnológico son:

- Lenguajes de maquina
- Lenguajes de bajo nivel (ensamblador)
- Lenguajes de alto nivel (lenguajes de programación)

Lenguajes de maquina

Son aquellos lenguajes que son comprendidos directamente por la computadora. Estas ultimas, en su capa de funcionamiento mas interna trabajan únicamente con unos (1) y ceros (0), y es de sistema binario de unos y ceros del cual se compone el lenguaje descrito. Con este se especifica desde una gran operación hasta las posiciones (dirección) de memoria necesarias para ejecutar dicha operación.

Este lenguaje, conocido también como código binario difiere en cada una de las computadoras que existan en el mundo. Esto se da puesto que dependiendo del hardware de la misma se obtendrán instrucciones con resultados mas o menos potentes, o igual de eficaces.

Se considera algo complejo trabajar con este tipo de lenguaje, sin embargo, una ventaja que destaca de su uso es el tener la posibilidad de transferir un programa a la memoria sin la necesidad de usar los **compiladores** o **traductores** mencionados anteriormente, que por ende supone un aumento de la velocidad de ejecución del mismo programa. Dicha velocidad es mayor comparada con el uso de programas traductores, es decir, manejando otro tipo de lenguajes.

Inconvenientes

En la actualidad, los inconvenientes superan las ventajas, lo que hace prácticamente no recomendable el lenguaje máquina.

Los inconvenientes son:

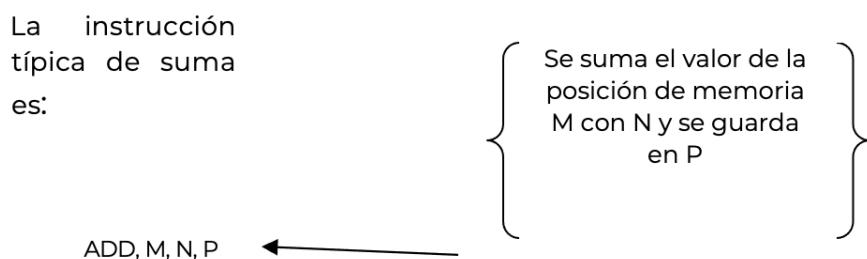
Introducción a la programación con C

- Dificultad y lentitud en la codificación.
- Poca confiabilidad.
- Dificultad grande de verificación y compilación de los programas.
- Los programas se ejecutan en un solo tipo de procesador (para el que fueron programados)

Lenguajes de bajo nivel

Estos son en teoría más fáciles de utilizar que los lenguajes máquina, pero al igual que ellos, dependen de la maquina en particular. El lenguaje de bajo nivel por excelencia es un lenguaje ensamblador. La sintaxis que compone las instrucciones en este lenguaje ensamblador son instrucciones conocidas como **nemotécnico**.

Ejemplo: nemotécnicos típicos de operaciones aritméticas son en inglés ADD, SUB, DIV en español RES, DIV, SUM.



Evidentemente, es mucho más fácil trabajar con instrucciones nemotécnicas que con código máquina. Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la computadora. Esto lo diferencia esencialmente con el lenguaje máquina.

El programa escrito en lenguaje ensamblador se conoce con el nombre de **programa fuente**, el programa traducido a lenguaje máquina se conoce con el nombre de **programa objeto**. El traductor del programa fuente a objeto es un programa llamado compilador.

Ventajas

- Mayor facilidad de codificación y en general su velocidad de cálculo.

Inconvenientes

- Dependencia por completo de la máquina impidiendo que el programa se pueda transportar y ejecutar en otras computadoras.
- La formación en lenguaje máquina es más compleja que la de lenguajes de alto nivel ya que exige no sólo la técnica de programación, sino también el conocimiento del interior de la máquina.
- “Tiene aplicaciones muy reducidas en la programación y se centra en aplicaciones de tiempo real, control de procesos y de dispositivos electrónicos.”(Aparicio Gil, Blanco Jiménez et all. TEMA 2: Lenguajes de programación)

Lenguajes de alto nivel

Estos lenguajes son los más utilizados en el ámbito de programación, están diseñados para que las personas **escriban** y **entiendan** los programas de un modo más sencillo. Otra razón es que estos lenguajes son independientes de la máquina, por lo que permiten generar programas transportables, teniendo la posibilidad de ser ejecutados con poca o ninguna modificación en diferentes maquinas.

Ventajas

- El tiempo de formación generalmente suele ser más corto.
- La sintaxis de los lenguajes de alto nivel se asemeja al lenguaje humano. Por ejemplo se podrían tener instrucciones con nombres como **READ, WRITE, PRINT, OPEN**.
- Las modificaciones y puesta a punto de los programas son más fáciles.
- Reduce el costo de los programas
- Transportabilidad.

Inconvenientes

- Incremento en las modificaciones del programa
- “No se aprovechan los recursos internos de la máquina, que se explotan mucho mejor en lenguaje máquina y ensamblador”.(Aparicio Gil, Blanco Jiménez et all. TEMA 2: Lenguajes de programación)
- Aumento de la ocupación de la memoria
- El tiempo de ejecución es mucho mayor.

Introducción a la programación con C

Los lenguajes de alto nivel existentes son muchos, aunque en la práctica su uso mayoritario se reduce a lenguajes como:

C, C++, Cobol, Fortran, Pascal, Visual Basic, Java (Uso extendido)
Prolog, Ada.95 (Gran uso en el mundo profesional)
Java, HTML, XML, Java Script, Visual j y últimamente PHP
(Lenguajes de Internet)

Los traductores de lenguaje

Son programas que traducen el código fuente escrito en lenguajes de alto nivel a código máquina.

Existen dos tipos de traductores:

- Compiladores
- Intérpretes

Intérpretes

Funciona como un traductor, toma un programa fuente, lo reescribe en lenguaje de máquina y luego lo ejecuta. Algunos programas intérpretes clásicos como **Basic** ya no son tan utilizados a día de hoy, aunque pueden encontrarse en computadoras viejas que funcionen con la versión **Qbasic**, bajo los sistemas operativos **DOS**, que corre en los computadores personales.

Compilador

Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación, a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un código intermedio (bytecode), o simplemente texto. Este proceso de traducción se conoce como compilación.

“Después de que se ha diseñado el algoritmo y escrito el programa en un papel, se debe comenzar el proceso de introducir el programa en un archivo en el disco duro de la computadora. La introducción y modificación de su programa en un archivo se hace utilizando un editor de texto o simplemente un editor.” (Aparicio Gil, Blanco Jiménez et all. TEMA 2: Lenguajes de programación)

Existen diferentes tipos de programas como:

- Programas de edición (texto, traductores)
- Programas científicos.
- Programas comerciales (contabilidad, análisis estadístico, nóminas, inventarios)

Los algoritmos

¿Qué es un algoritmo?

Es un conjunto finito de reglas que dan una secuencia de operaciones para resolver todos los problemas de un tipo dado. De forma más sencilla, podemos decir que un algoritmo es un conjunto de pasos que nos permiten obtener un dato. Además, un algoritmo debe cumplir estas condiciones:

- **Finitud:** el algoritmo debe acabar tras un número finito de pasos. Es más, es casi fundamental que sea en un número razonable de pasos.
- **Definibilidad:** el algoritmo debe definirse de forma precisa para cada paso, es decir, hay que evitar toda ambigüedad al definir cada instrucción. Puesto que el lenguaje humano es impreciso, los algoritmos se expresan mediante un lenguaje formal, ya sea matemático o de programación para un computador.
 - **Entrada:** el algoritmo tendrá cero o más entradas, es decir, cantidades dadas antes de empezar el algoritmo. Estas cantidades pertenecen, además, a conjuntos especificados de objetos. Por ejemplo, pueden ser cadenas de caracteres, enteros, naturales, fraccionarios, etc. Se trata siempre de cantidades representativas del mundo real, expresadas de tal forma que sean aptas para su interpretación por el computador.
 - **Salida:** el algoritmo tiene una o más salidas, en relación con las entradas.
 - **Efectividad:** se entiende por esto que una persona sea capaz de realizar el algoritmo de modo exacto y sin ayuda de una máquina en un lapso de tiempo finito.

A menudo, los algoritmos requieren una organización bastante compleja de los datos, y es necesario un estudio previo de las estructuras de datos fundamentales. Dichas estructuras pueden implementarse de diferentes maneras, y, es más, existen algoritmos para implementar dichas estructuras. El uso de estructuras de datos adecuadas puede hacer trivial el diseño de un algoritmo, o un algoritmo muy complejo puede usar estructuras de datos muy simples.

Uno de los algoritmos más antiguos conocidos es el algoritmo de Euclides. El término algoritmo proviene del matemático Muhammad

ibn Musa al-Khwarizmi, que vivió aproximadamente entre los años 780 y 850 D.C. en la actual nación iraní. Él describió la realización de operaciones elementales en el sistema de numeración decimal. De al-Khwarizmi se obtuvo la derivación de algoritmo.

Clasificación de algoritmos

-**Algoritmo determinista:** ese determina de forma única el siguiente paso al momento de su ejecución.

-**Algoritmo no determinista:** en cada paso de la ejecución se debe decidir de entre varias alternativas, todas se deben agotar para encontrar la solución.

Todo algoritmo tiene una serie de características, entre otras, que requiere una serie de recursos, algo que es fundamental considerar a la hora de implementarlos en una máquina, estos recursos son principalmente:

-**El tiempo:** período transcurrido entre el inicio y la finalización del algoritmo.

-**La memoria:** la cantidad (la medida varía según la máquina) que necesita el algoritmo para su ejecución.

Obviamente, la capacidad y el diseño de la máquina pueden afectar al diseño del algoritmo.

-**En general,** la mayoría de los problemas tienen un parámetro de entrada que es el número de datos que hay que tratar, esto es, N . La cantidad de recursos del algoritmo es tratada como una función de N . De esta manera puede establecerse un tiempo de ejecución del algoritmo que suele ser proporcional a una de las siguientes funciones:

-**Tiempo de ejecución constante:** Significa que la mayoría de las instrucciones se ejecutan una o muy pocas veces.

-**Log N:** Tiempo de ejecución logarítmico. Se puede considerar como una gran constante. La base del logaritmo (en informática la más común es la base 2) cambia la constante, pero no de forma exagerada. El programa es más lento cuanto más crezca N , pero es inapreciable, pues $\text{Log } N$ no se duplica hasta que N llegue a N^2 .

-**N:** Tiempo de ejecución lineal. Un caso en el que N valga 40, tardará el doble que otro en el que N valga 20. Un ejemplo sería un algoritmo que lee N números enteros y devuelve la media aritmética.

-**N Log N:** El tiempo de ejecución es $N \text{ Log } N$. Es común encontrarlo en algoritmos. Si N se duplica, el tiempo de ejecución es ligeramente mayor del doble.

Introducción a la programación con C

-**N²:** Tiempo de ejecución cuadrático. Suele ser habitual cuando se tratan pares de elementos de datos, como por ejemplo un bucle anidado doble. Si N se duplica, el tiempo de ejecución aumenta cuatro veces.

-**N³:** Tiempo de ejecución cúbico. Como ejemplo se puede dar el de un bucle anidado triple. Si N se duplica, el tiempo de ejecución se multiplica por ocho.

-**2^N:** Tiempo de ejecución exponencial. No suelen ser muy útiles en la práctica por el elevadísimo tiempo de ejecución. Si N se duplica, el tiempo de ejecución se eleva al cuadrado.

-**Algoritmos polinomiales:** aquellos que son proporcionales a **N^k**. Son en general factibles.

-**Algoritmos exponenciales:** aquellos que son proporcionales a **k^N**. En general son no factibles salvo un tamaño de entrada muy reducido.

Pasos para diseñar un programa: Forma 2

Aunque el proceso de diseñar programas es esencialmente un proceso creativo, se pueden considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

Las siguientes son las etapas que se deben cumplir para resolver con éxito un problema de programación:

- Definición del problema
- Análisis del problema
- Selección de la mejor alternativa
- Diagramación
- Prueba de escritorio
- Codificación
- Transcripción
- Compilación
- Pruebas de computador
- Documentación externa

1. Definición del problema

Está dada por el enunciado del problema, el cuál debe ser claro y completo, es importante que se conozca exactamente qué se desea del computador. Mientras que esto no se comprenda, no es posible pasar a la siguiente etapa.

2. Análisis del problema

Entendido el problema ¿qué se desea obtener del computador? Para resolverlo es preciso analizar:

- Los datos o resultados que se esperan.
- Los datos de entrada que son entregados.
- El proceso por el cual se someterán esos datos con el objetivo de obtener ciertos resultados como valores (numéricos, de texto), fórmulas, entre otros.

Una recomendación muy útil es que el programador se ponga en el lugar del computador y analice que necesitaría que le ordenen y en qué secuencia, para poder generar los resultados esperados.

3. Selección de la mejor alternativa

Analizado el problema, puede que existan múltiples formas de resolverlo, aquí también es válido el principio de que las cosas siempre se podrán hacer de una mejor forma. Lo importante es determinar cuál es la mejor alternativa, esta es aquella que genera los resultados esperados en el menor tiempo y al menor costo.

4. Diagramación

Teniendo en mente una opción de cómo resolver el problema, se procede a graficar la lógica de dicha opción. El gráfico viene siendo lo que se conoce como Diagrama de Flujo: la representación gráfica de una secuencia lógica de pasos a cumplir por el computador para producir un resultado esperado.

La experiencia demuestra, que resulta muy útil trasladar los pasos propuestos en el diagrama a frases que indiquen lo mismo; es decir, hacer una codificación del programa, pero utilizando instrucciones en español. Como si se le estuviera hablando al computador, esto es lo que se denomina Algoritmo o pseudocódigo.

Cuando el programador adquiera mas experiencia desarrollando programas, es posible que no necesite elaborar el diagrama de flujo, en su lugar puede proceder a hacer directamente el pseudocódigo del programa.

5. Prueba de escritorio

Para garantizar que los diagramas y pseudocódigos además del programa codificado funcione correctamente, es conveniente someterlos a pruebas de escritorio. Esta prueba consiste en establecer diferentes datos de entrada al programa (en cualquiera de sus representaciones) y seguir la secuencia de pasos lógicos creados, para que al final del recorrido se verifique si los resultados son los

Introducción a la programación con C

esperados. De los pasos anteriores se define si la lógica utilizada es correcta o si hay necesidad de hacer ajustes (volver al paso 4). Se realizar mas de una prueba de escritorio y considerar todos los casos posibles, incluyendo los de excepción o no esperados, para poder asegurar que el programa no generará errores en ejecución.

6.Codificación

Una vez que se verificado el programa usando las pruebas de escritorio, se procede codificar el programa en el lenguaje de computador seleccionado. Esto es, reescribir los pasos del diagrama en instrucciones que utilizan un lenguaje que el computador reconoce.

Todos los lenguajes de programación proporcionan las facilidades para incluir líneas de comentarios en el desarrollo de los programas. Estos comentarios facilitan entender el programa. Resulta muy conveniente su uso debido a que estos no son tenidos en cuenta por el computador a la hora de ejecutar el programa. Por otra parte, estos comentarios se denominan a nivel general como Documentación Interna.

7.Transcripción

Es necesario que se lleve el programa codificado a un medio que sea aceptado como entrada por el computador: perforado en tarjetas, grabado en un disco flexible o grabado en un disco duro. Este programa es el que se conoce como Programa Fuente.

8.Compilación

Se procede a utilizar a continuación un “programa de computador llamado Compilador o Traductor, el cuál analiza todo el programa fuente y detecta errores de sintaxis ocasionados por fallas en la codificación o en la transcripción. Las fallas de lógica que pueda tener un programa fuente no son detectadas por el compilador. Cuando no hay errores graves en la compilación, el compilador traduce cada instrucción del programa fuente a instrucciones propias de la máquina (Lenguaje de Maquina), creando el Programa Objeto.

Algunos computadores utilizan Interpretadores, (Generalmente para el Lenguaje Basic), en reemplazo de programas compiladores. La diferencia consiste en que el interpretador recibe desde una terminal sólo una instrucción a la vez, la analiza y si está bien la convierte al formato propio de la máquina. Si la instrucción tiene algún error, el interpretador llama la atención de la persona para que

Introducción a la programación con C

corrija dicha instrucción"(M. S. Zigor Aldazabal, Instituto Específico Formación Profesional Superior "BIDASOA", PROGRAMACIÓN EN C++ Y VISUAL BASIC, 2009) .

Como resultado de la corrida del compilador, podemos obtener varios listados:

- Listado del programa fuente
- Listado de los errores detectados
- Listado de campos utilizados, etc.

Los errores se deben corregir sobre el mismo programa fuente, ya sea reemplazando las tarjetas mal perforadas o regrabando en el disco flexible o disco duro. Este paso de la compilación se repite hasta eliminar todos los errores y obtener el programa ejecutable.

9.Pruebas de computador

Cuando se tiene un programa ejecutable (en lenguaje de máquina), se le ordena al computador que lo ejecute, para lo cual se ingresan datos de prueba, como se hizo en la prueba de escritorio (paso 5). Los resultados obtenidos se analizan, luego de lo cual puede ocurrir cualquiera de estas situaciones:

-La lógica del programa está bien, pero existen algunos errores sencillos de solucionar modificando, añadiendo o eliminando instrucciones. Este proceso se repite hasta el paso 6.

-Existen errores que son resultados de la lógica utilizada, obligando al programador a regresar a los pasos 4 y 5 para revisión y modificación del diagrama.

-Hay errores muy graves y lo más aconsejable es regresar al paso 2 para analizar nuevamente el problema y repetir todo el proceso.

-No existen errores y los resultados son los esperados. En este caso, el programa se puede guardar permanentemente en una librería o biblioteca del computador, para sacarlo de allí cuando se necesite ejecutar nuevamente.

10.Documentación externa

Es conveniente hacer la documentación externa de un programa luego de que este se da por finalizado. En esta documentación se podría indicar las normas de la instalación, recomendaciones para futuros cambios, y en general cualquier otra información que pueda

Introducción a la programación con C

aportar a una clara comprensión del programa. Por otra parte , los elementos que no pueden faltar en una buena documentación son:

- Enunciado del problema
- Diagrama de pasada
- Narrativo con la descripción de la solución
- Relación de las variables o campos utilizados en el programa, cada uno con su respectiva función
- Diagrama del programa
- Listado de la última compilación
- Resultados de la ejecución del programa.

Algoritmo o pseudo código

Tipos de algoritmos

Existen dos tipos de algoritmos:

- Los algoritmos informales son aquellos que se realizan por medio de palabras sencillas tal como se comunican las personas comúnmente.
- Los algoritmos formales son los que se pueden resolver mediante el uso de unas instrucciones.

Ejemplo 1: Diseñar un algoritmo para preparar una limonada.

Nombre _algoritmo Limonada

INICIO

PASO 1 Llenar una jarra con un litro de agua

PASO 2 Echar el jugo de tres limones

PASO 3 Echar cuatro cucharadas de azúcar

PASO 4 Revolver el agua hasta disolver completamente el azúcar
FIN

Ejemplo 2: Diseñar un algoritmo que permita hallar la suma y el promedio de tres números.

INICIO

PASO 1 leer numero1, numero2, numero3

PASO 2 sumar = numero1 + numero2 + numero3

PASO 3 promedio = suma / 3

PASO 4 imprimir suma, promedio

FIN

Notas:

- El término LEER significa obtener un dato de algún dispositivo de entrada como el teclado y almacenarlo en una variable.

Introducción a la programación con C

-Una variable es una localización en la memoria que tiene un nombre y cuyo contenido puede cambiar a lo largo de la ejecución de un programa. Así numero1, numero2 y numero3 son variables.

-El término IMPRIMIR significa mostrar el valor de una variable en algún dispositivo de salida, como la pantalla.

Ejemplo 3: Diseñar un algoritmo para freír un huevo.

INICIO

PASO 1 Buscar ingredientes

PASO 2 Sí encuentro todos los ingredientes

PASO 3 Prender estufa

PASO 4 Poner a derretir la mantequilla

PASO 5 Picar ingredientes

PASO 6 Echar ingredientes al sartén

PASO 7 Esperar que esté listo

PASO 8 Apagar estufa

PASO 9 Servir huevo

FIN

Ejercicios sobre algoritmos informales

- Desarrollar un algoritmo que permita tomar un avión
- Desarrollar un algoritmo que permita manejar un auto
- Desarrollar un algoritmo que permita seleccionar un tema
- Desarrollar un algoritmo que permita buscar un libro
- Desarrollar un algoritmo que permita hacer un pastel.
- Desarrollar un algoritmo que permita dar un beso.
- Desarrollar un algoritmo que permita maquillarse.
- Desarrollar un algoritmo que permita tomar una fotografía
- Desarrollar un algoritmo que permita cambiar una llanta.
- Desarrollar un algoritmo que permita cambiar un bombillo.
- Desarrollar un algoritmo que permita ir a cine.
- Desarrollar un algoritmo que permita cambiar un cuadro
- Desarrollar un algoritmo que permita grabar un disco.
- Desarrollar un algoritmo que permita presentar un examen
- Desarrollar un algoritmo que permita freír un huevo
- Desarrollar un algoritmo que permita pintar una casa
- Desarrollar un algoritmo que permita seleccionar una camisa
- Desarrollar un algoritmo para colocarnos una camisa.

Algoritmos formales

El primer programa en pseudocódigo

Normas:

- Comienza con el nombre del programa
 - Definición de las variables globales y locales
 - Todos los programas comienzan por la palabra Inicio, que da apertura al programa y a los procesos correspondientes, seguido de un grupo de sentencias que mecánicamente están organizadas para dar viabilidad, cohesión al desarrollo de este y de esta manera poder recibir los datos deseados.
 - Por tanto, debemos recordar que un grupo de sentencias va entre llaves {...}, las cuales las agrupa para hacer referencia a:
 - El hecho de que son varias implica su agrupamiento.
 - Un ordenamiento sintáctico.
 - Fácil compresión de las líneas de código.
 - Las sentencias tienen dependencia mutua para arrojar uno o más resultados, como se verá más adelante en Estructuras de Control Repetitivas.
 - Una sentencia simple aislada no necesita llaves.
 - Generalmente se usa el carácter de punto y coma ';' (**Separador de sentencias**). Este indica el final de una sentencia
 - Como norma se declaran todas las variables al inicio del programa y se les asigna el valor correspondiente dependiendo del tipo (entero, real, carácter) si es necesario. **Ejemplo:**
- Formato** entero s;

Instrucciones para entrada y salida de datos

Se tiene la expresión **escribir()** o **escribe()**. Estas presentan en la pantalla estándar (hablando de un computador) las expresiones, valores, variables y/o cadenas contenidas en las expresiones anteriores, por **ejemplo**:

escribe ("Introducir el valor de la base"); //muestra en pantalla la cadena contenida.

escribe ("El resultado es", p); //imprime un mensaje y el valor que contiene la variable p de tipo entero.

escribe (p); // solamente imprime el valor de la variable p.

Se requiere de una sintaxis especial si se desea presentar las expresiones de manera conjunta.

Cadenas y variables

Se escribe la cadena y luego de una coma se indica la variable. Se continúa de esta manera hasta representar por completo el mensaje con las variables necesarias.

Pseudocódigo:

escribe ("base", b, "por la altura", a, "es la superficie", s);
Lenguaje c

```
printf("base %d", b, "por la altura %d", a, "es la superficie %d", s)
```

Tipos de formatos

Manejo de datos numéricos:

Identificador	Descripción
%d	Para el manejo de números enteros
%i	Para el manejo de números enteros
%f	Para el manejo de números reales
%o	Para el manejo de números en Octal
%x	Para el manejo de números en Hexadecimal
%p	Para el manejo de apuntadores

Manejo de caracteres y cadenas:

Identificador	Descripción
%s	Para el manejo de cadenas
%c	Para el manejo de caracteres

Introducción a la programación con C

Manejo de presentación por pantalla:

Identificador	Descripción
%v	Para visualizar el texto o números verticalmente
%h	Para visualizar el texto o números horizontalmente

Ejemplo:

```
//Visualiza un mensaje por pantalla:  
printf("hola como estas");  
int A=1;  
//Visualiza un mensaje y el contenido de la variable A de tipo entero:  
printf("el valor es %d", A);
```

Instrucciones de Entrada

Según Machaca M. en su escrito *Introducción a l programación en visual Basic .NET 2008, 2014* (Cap. 9-12) las instrucciones leer () y **scanf ()** toman los valores desde el dispositivo de entrada como lo es el teclado, luego los introduce en las variables que formen parte del uso de la instrucción. Los datos introducidos deben ser del mismo tipo que el de las variables que los almacenan. Hay dos formas de usarlas:

Formato pseudocódigo;

```
...  
x = leer ();  
leer(x);
```

Cuando se quiere recoger más de un dato se utiliza esta forma:

leer (x, y)

Se respetará el orden de recogida con cual, si se introduce desde el teclado 12 y 34, la variable x =12 y la y =34.

Formato scanf

```
scanf ("tipo formato", dirección Nom_Variable);  
scanf ("%d", &h);
```

La dirección indica la posición donde queda almacenada la variable. En este caso la variable h

Introducción a la programación con C

Ejemplo:

```
int h;  
printf("Digite un número");  
scanf("%d",&h)
```

Suponga que se quiere hacer un programa que calcule el área de un triángulo. Primero se puede tener en cuenta que el área mencionada se expresa con la siguiente formula $A = (b*h) / 2$, donde las variables **A**, **b**, **h** seguidamente representan **área**, **base** y **altura**. Por lo tanto, se deben pedir dos datos y presentar un resultado.

Una posible solución usando una secuencia de sentencias y expresiones sería:

<u>OPERADOR ARITMÉTICO</u>	<u>NOMBRE</u>	<u>UTILIZACIÓN</u>	<u>RESULTADO</u>
+	Suma	Op1+op2	Op3
-	Resta	Op1-op2	Op3
*	Multiplicación	Op1*op2	Op3
/	División	Op1/op2	Op3
%	Modulo o resto de la división	Op1%op2	0

<u>OPERADORES INCREMENTALES</u>	<u>NOMBRE</u>	<u>UTILIZACIÓN</u>	<u>RESULTADO EQUIVALENTE</u>
++	Incremento.	++ i	i + 1
--	Decremento.	--i	i - 1
=	Multiplicación	Op1= Op2	Op1= op1 * op2
/=	División	Op1/= Op2	Op1= op1 / op2
%=	Modulo o resto de la división	Op1%= Op2	Op1= op1 % op2

<u>OPERADORES RELACIONALES</u>	<u>NOMBRE</u>	<u>UTILIZACIÓN</u>	<u>RESULTADO</u>
>	Mayor que	Op1 > Op2	Si op1 es mayor que op2

Introducción a la programación con C

<	Menor que	$Op1 < Op2$	Si op1 menor que op2
<=	Menor o igual que	$Op1 <= Op2$	Si op1 menor o igual que op2
>=	Mayor o igual que	$Op1 >= Op2$	Si op1 mayor o igual a op2
==	igual	$Op1 == Op2$	Si op1 es igual a op2
!=	diferente	$Op1 != Op2$	Si op1 es diferente a op2

<u>OPERADORES LÓGICOS</u>	<u>NOMBRE</u>	<u>UTILIZACIÓN</u>	<u>RESULTADO</u>
&&	AND	$Op1 \&\& Op2$	True (verdadero) si op1 y op2 son true. Sin op1 es false (falso) ya no será necesario evaluar op2 ya que esta condición no se cumple.
 	OR	$Op1 Op2$	True (verdadero) si op1 u op2 son true. Sin op1 es true ya no será necesario evaluar op2 ya que este operador es aceptado si por lo menos una condición es verdadera.
!	NEGACIÓN	$! Op$	Este se encarga de tomar el operador de forma contradictoria, es decir si op es true el aceptará a op como false y viceversa.
&	AND	$Op1 \& Op2$	True si op1 y op2 son true. Siempre se evalúa op2.
 	OR	$Op1 Op2$	True si op1 u op2 son true. Siempre se evalúa op2.

Introducción a la programación con C

<u>OPERADORES QUE ACTÚAN AL NIVEL DE BITS</u>		<u>UTILIZACIÓN</u>	<u>RESULTADO</u>
<<		Op1 << Op2	Desplaza los bits de op1 a la derecha una distancia op2
>>		Op1>> Op2	Desplaza los bits de op1 a la izquierda una distancia op2.
>>>		Op1 >>> Op2	Desplaza los bits de op1 a la derecha una distancia op2 (Positiva).
&		Op1 & Op2	Operador AND a nivel de bits.
		Op1 Op2	Operador OR a nivel de bits.
^		Op1 ^ Op2	Operador XOR a nivel de bits (1 si solo uno de los operadores es 1).
~		~ Op2	O p e r a d o r Complemento (invierte el valor de cada bit).

Pseudocódigo

Área del triángulo // nombre del programa

Inicio Enteros a, b, h; // Declaramos las variables

Escribe("Introducir el valor de la Base y altura."); //mensaje en pantalla
Leer (b); // Leemos la base Leer (h); // Leemos la altura a = (b * h)/2; //
calculamos //Presentamos el resultado en la misma línea del mensaje
Escribe ("la superficie es: ", a); Fin

Lenguaje C

```
1 main() {  
2     int a, b, h; // Declaro las variables  
3     //mensaje en pantalla  
4     printf("Introducir el valor de la Base  
y altura: ");  
5     scanf ("%d", b); // Leemos la base  
6     scanf ("%d", h); // Leemos la altura  
7     a = (b * h)/2; // calculamos...  
8     //Presentamos el resultado:  
9     printf ("la superficie es : %d", a);  
10 }
```

Ejemplo de Algoritmo formales

Algoritmo Área _ triangulo	1. Nombre Algoritmo
Inicio	2. Inicio del algoritmo
escriba (" la base y la altura ") leer: base, altura	3. Datos de entrada
area= base * altura/2	4. Proceso
escriba ("El área del triángulo es ", area)	5. Datos de salida
Fin	6. Final del algoritmo

Algunos ejemplos de algoritmos son: las recetas de cocina, instrucciones para cambiar una llanta, etc.

Problemas de pseudocódigo

Declaración de Variables y Programas secuenciales

1. Decir si son correctos o no los siguientes identificadores de variables:

Variables	Correcto	Incorrecto
Contad		
N_Bytes		
2oral		
3M_Atrim		
&6Alor"		
6"#\$9i		
#32gt		

2. Escribir el pseudocódigo de un programa que calcule de forma individual la velocidad de 4 cuerpos, introduciendo por teclado el espacio y el tiempo, posteriormente se debe imprimir por pantalla los tres valores mencionados (distancia, tiempo y velocidades calculadas).

Pseudocódigo

Inicio

```
real: e1, e2, e3, e4, t1, t2, t3, t4, v1, v2, v3, v4;  
real: espacio_total, tiempo_total, velocidad_total;  
escribe ("digite la distancia cuerpo 1"); leer(e1);  
escribe ("digite tiempo del cuerpo 1"); leer(t1);  
escribe ("digite distancia cuerpo 2"); leer(e2);  
escribe ("digite tiempo del cuerpo 2"); leer(t2);  
escribe ("digite distancia cuerpo 3"); leer(e3);  
escribe ("digite tiempo del cuerpo 3"); leer(t3);  
escribe ("digite distancia cuerpo 4"); leer(e4);  
escribe ("digite tiempo del cuerpo 4"); leer(t4);  
v1 = e1 / t1;  
v2 = e2 / t2;  
v3 = e3 / t3;  
v4 = e4 / t4;  
espacio_total = e1+e2+e3+e4;  
tiempo_total = t1+t2+t3+t4;  
velocidad_total = v1+ v2+ v3+ v4;  
escribe ("velocidad del cuerpo 1:", v1);  
escribe ("velocidad del cuerpo 2:", v2); escribe ("velocidad del cuerpo 3:", v3);  
escribe ("velocidad del cuerpo 4:", v4);  
escribe ("velocidad total de los cuatro cuerpos:", velocidad_total);  
escribe ("espacio total de los cuerpos:", espacio_total);  
escribe ("tiempo total de los cuerpos:", tiempo_total);
```

Fin

- 3.** Realizar un programa en pseudocódigo que calcule el perímetro y el área total de tres circunferencias sabiendo que la 1^a de ellas tiene radio R, la 2^a tiene radio 2R y la 3^a tiene radio 3R. El radio de la 1^a circunferencia será introducido por teclado.

Pseudocódigo

Inicio

```
real: r, pi, a1, a2, a3, p1, p2, p3, area_total, perímetro_total;  
escribe ("Digite el valor del radio: "); leer (r);  
escribe ("Circunferencia 1:");  
pi = 3.1416; a1= pi * (r * r);  
p1 = 2 * (pi * r);  
escribe("Su perímetro es: ", p1);  
escribe("Su área es: ", a1);  
escribe("Circunferencia 2:");
```

Introducción a la programación con C

```
r = 2 * r; a2 = pi * (r * r);
p2 = 2 * (pi * r);
escribe("Su perímetro es: ", p2);
escribe("Su área es: ", a2);
escribe("Circunferencia 3");
r = 3 * r; a3 = pi * (r * r); p3 = 2 * ( pi * r);
escribe("Su perímetro es: ", p3);
escribe("Su área es: ", a3);
perímetro_total = p1+ p2+ p3;
area_total = a1+ a2+ a3;
escribe("Perímetro total de las circunferencias:", perímetro_total);
escribe("Área total de las circunferencias:", area_total);
Fin
```

Ejercicios del tema 1: diseño de algoritmos simples

En los siguientes ejercicios se pretende descubrir un algoritmo para solucionar distintos problemas, de forma que sólo se empleen instrucciones elementales: Operaciones aritméticas (suma, resta, multiplicación...), asignación, entrada, salida, bucles y selección.

Para cada ejercicio intente examinar cómo se efectúa la operación 'a mano'. Se trata de ver que cuando se soluciona un ejercicio, también se soluciona ejecutando distintas operaciones una a una. Despues, trate de descubrir las operaciones repetitivas, que serán los bucles y las operaciones en las que pregunta por una condición (Si se cumple que... entonces... y si no...), que serán operaciones de selección. Por último, construya el diagrama de flujo siguiendo una forma de representación de algoritmos.

Ejercicios

1.Diseñe un diagrama de flujo que halle el área y el perímetro de un rectángulo. Considere las siguientes fórmulas: área = base x altura, perímetro = 2 x (base + altura).

2.Diseñe un diagrama de flujo para convertir una longitud dada en centímetros a pies. Considere que:

1 pie = 30.48 centímetros.

3.Diseñar un algoritmo que lea 3 números enteros positivos distintos e imprima el mayor valor de los 3. Supondremos que los 3 números que se leen son distintos.

4.Efectuar un algoritmo que averigüe si un número positivo es par o impar. El programa leerá un número entero positivo y dará el resultado.

5.Diseñar un algoritmo para sumar los 100 primeros números naturales. El algoritmo no leerá ningún valor. Simplemente mostrará el resultado.

Introducción a la programación con C

6.Diseñar un algoritmo para convertir temperaturas en grados Fahrenheit a grados Celsius. El programa terminará cuando lea una temperatura igual a 999. La relación entre grados Fahrenheit (F) y grados Celsius (C) viene dada por: $C = \frac{5}{9}(F - 32)$.

7.Diseñar un algoritmo que lea 2 puntos en el plano (4 números reales): (X_1, Y_1) y (X_2, Y_2) y devuelva la distancia euclídea entre ellos. Para ello se usará el teorema de Pitágoras.

8.El sueldo neto de un vendedor se calcula como la suma de un sueldo básico de \$./250 más el 12% del monto total vendido. Diseñe un algoritmo que determine el sueldo neto de un vendedor sabiendo que hizo tres ventas en el mes.

9.Escribir un diagrama de flujo para un algoritmo que calcule la media de varios números, los que introduzca el usuario por teclado, y saque el resultado por pantalla. Nota: el primer carácter no numérico que introduzca el usuario indicará que no va a introducir más números. Usar tres variables: contador, suma y resultado en expresiones aritméticas sencillas. Considerar primitivas las acciones de leer por teclado y escribir en pantalla.

10.Escribir algoritmos en pseudocódigo que calculen y saquen por pantalla el máximo de tres números introducidos por teclado.

11.Dados X y Y , verificar si X es divisible por Y .

12.Sumar los cuadrados de los primeros N números naturales.

13.Leer N números y obtener el promedio solo de los números pares de la lista.

14.Introducir un conjunto de M números, determinar la cantidad de números positivos y negativos del conjunto.

Realizar el diagrama de flujo de los siguientes problemas

1.Realizar un programa que compare dos números ingresados por teclado, el resultado del programa debe ser correcto, si los números son iguales, de lo contrario debe sacar un mensaje especificando que no son iguales y escribir si es mayor o menor que el número comparado.

2.Generar los números pares e impares de 40 hasta 400.

3.Realizar un algoritmo para vigilar una ventana las veinticuatro horas.

4.Realizar un algoritmo que lea 3 números enteros y mostrarlos ascendenteamente.

5.Realizar un programa que lea 4 números enteros por teclado y saque el menor y mayor de los cuatro números.

6.Realizar un programa que lea n números y saque la cantidad de números positivos y negativos que existen utilizando un ciclo infinito,

7.Leer dos números enteros y mostrar todos los números comprendidos entre ellos.

Introducción a la programación con C

8.Leer una serie de números enteros hasta que digite el número 99 y determinar La cantidad de número que ingreso por teclado.

9.La sumatoria de números pares positivos.

10.La sumatoria de números impares negativos y la cantidad de ceros.

11.Mostrar la tabla de multiplicar del 1 hasta 13.

12.Mostrar la tabla de multiplicar de un número dado.

13.Mostrar los primeros cien números múltiplos de 3.

14.Leer dos números y mostrar todos los números múltiplos de cinco comprendidos entre ellos.

15.Realizar un programa que realice la siguiente conversión de pies a metro y de metros a pies

16.Escriba un programa que realice las siguientes conversiones

17.Programa que encuentra el mayor, el del medio y menor de tres números

a.metros a kilómetros

b.kilómetros a metros

c.decímetros a hectómetros

d.decímetros a milímetros

e.hectómetros a decímetros o milímetros

18.Utilizando ciclos anidados generar las siguientes parejas ordenadas

a) 0123456789 b) 012345678

1122334455 110001001

c) 123456789

1.1222333

19.Realizar un programa que determine la sumatoria de la siguiente serie

a) 2 ,5, 7, 10 ,12 ,17, ... ,1800

Capítulo 2

Capítulo 2: Programación en C

En este capítulo se dará una introducción a la programación en C.

Se verán conceptos acerca de la entrada y salida por pantalla de variables y mensajes, además de sintaxis básicas para crear un programa principal, algoritmos de ejemplo, bloques condicionales, y demás conceptos que ayudarán a entender mejor la programación en lenguaje C

Programación en C

Historia del lenguaje C

C es un lenguaje equilibrado de propósitos generales que se desarrolló a partir de estas raíces. Su definición aparece en 1978 en "C Reference Manual" del libro "The C Programming Language", de Brian W. Kernighan y Dennis M. Ritchie, pero el estándar recomendable más reciente apareció en junio de 1983, en el documento de los Laboratorios Bell titulado The C Programming Language-Reference Manual, escrito por Dennis M. Ritchie.

El lenguaje C está inspirado en el lenguaje B, escrito por Ken Thompson en 1970 con intención de recodificar UNIX, que en la fase de arranque está escrito en assembler, debido a su transportabilidad a otras máquinas.

B era un lenguaje evolucionado e independiente de la máquina, inspirado en el lenguaje BCPL concedido por Martin Richard en 1967.

En 1972, Dennis Ritchie, toma el relevo y modifica el lenguaje B, creando el lenguaje C y rescribiendo el UNIX en dicho lenguaje. La novedad que proporcionó el lenguaje C sobre el B fue el diseño de tipos de estructura de datos.

AT&T lo enseñó como un compilador (comp. C) llamado K&R C que junto con el sistema operativo UNIX empezaron a invadir universidades. Después, cada persona que adquiría una copia de UNIX recibía un compilador de C gratis. El lenguaje más popular fue C, por lo tanto UNIX fue escrito en C. Entonces si una persona quería entender UNIX, tenía que aprender C.

La característica principal era su gratuidad y entonces nadie se sentía presionado a aprenderlo. ¿Cuál fue el resultado?, un gran estándar.

Luego que C se convirtió en un gran estándar, entonces las compañías introducían sus propios compiladores C. Añadiendo la capacidad de ejecutarse en otros sistemas operativos que no fuesen UNIX.

Cada uno de estos compiladores introducía ensanchamientos diseñados para mejorar las limitaciones que mostraba el modelo original. Pero las modificaciones que cada quien hacía, traían como resultado la incompatibilidad de las versiones entre sí. Gracias a esto incrementó la demanda por un estándar a nivel nacional, entonces en 1987 nació el primer estándar "The American National Standards

Introducción a la programación con C

Institute (ANSI) version of C" esta versión fue mejor conocida como ANSI C o C estándar.

El C puede ser compilado al lenguaje de máquina en casi todas las computadoras. Por ejemplo el Unix está escrito en C, se ejecuta y compila en una amplia variedad de micro, mini y macrocomputadoras.

En C se programa con una serie de funciones que se llaman unas a otras para el procesamiento. Aun el cuerpo del programa es una función denominada flexible, permitiendo a los programadores la elección entre el uso de la biblioteca estándar que se provee con el compilador, el uso de funciones de terceros creadas por otros proveedores de C o el desarrollo de sus propias funciones.

Comparado con otros lenguajes de programación de alto nivel C parece complicado, su apariencia intrincada se debe a su extrema flexibilidad.

Las principales características de este lenguaje son:

- Programación ordenada.
- Facilidad de aprendizaje.
- Economía de sus expresiones.
- Gran cantidad en operadores y tipos de datos.
- Codificación en alto y bajo nivel simultáneamente.
- Reemplaza ventajosamente la programación en Asembler.
- Utilización natural de las funciones primitivas del sistema.
- No está orientado a ningún área en especial.

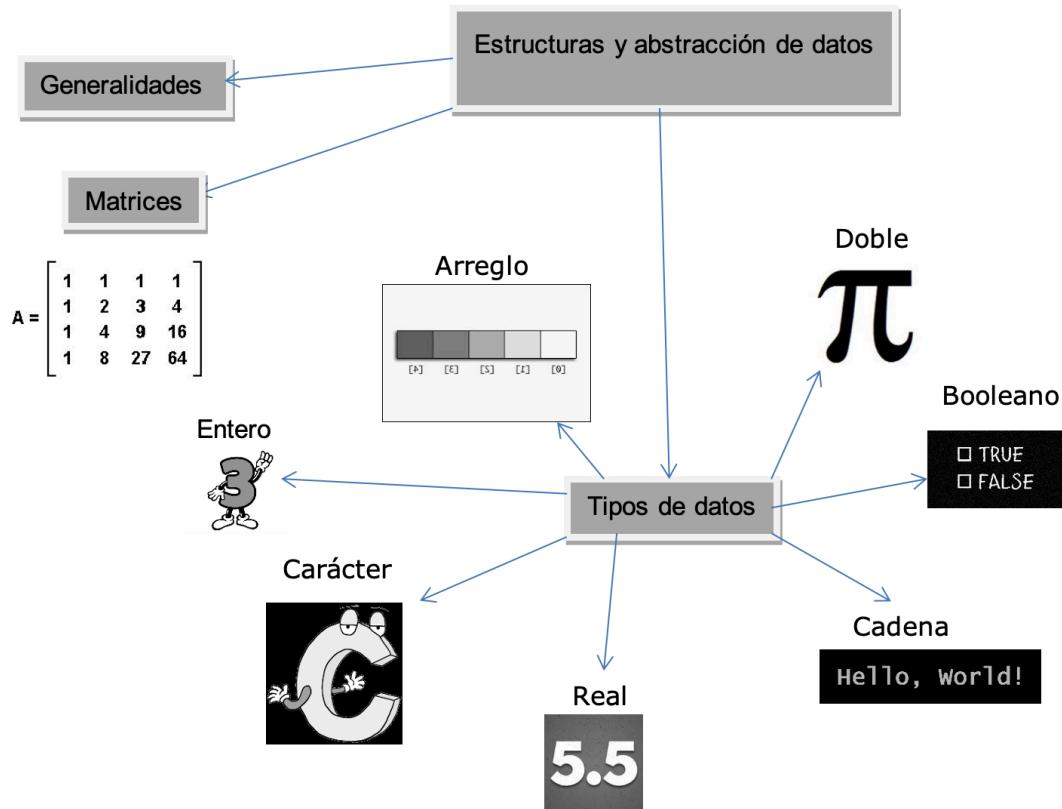
Hay toda una serie de operaciones que puede hacerse con el lenguaje C que realmente no están incluidas en el compilador propiamente dicho, sino que las realiza un preprocesador justo antes de cada compilación. Los dos más importantes son #define (directriz de sustitución simbólica o de definición) e #include (directriz en el fichero fuente).

Finalmente C ha sido pensado para ser altamente transportable y para programar lo improgramable, tiene algunos pequeños inconvenientes como todos los lenguajes:

Carece de instrucciones de entrada / salida, de instrucciones para manejo de cadenas de caracteres, con lo que este trabajo queda para la librería de rutinas, con la siguiente pérdida de transponibilidad.

La excesiva libertad en la escritura de los programas puede llevar a errores en la programación que por ser correctos sintácticamente no se detectan a simple vista.

Tipos de datos básicos en lenguaje c



Entero

Los números enteros son todos aquellos números que solo tienen una parte entera y **no tienen parte decimal, hacen parte del subconjunto finito de los números Naturales**, el rango depende de cuantos bits utilice para codificar el número, normalmente 2 bytes para números positivos.

Con 16 bits se pueden almacenar $2^{16} = 65536$ números enteros diferentes que van de 0 al 65535 y de -32768 al 32767 para números con signo.

Por ejemplo

2, 14, -199, ...

Operaciones asociadas al tipo entero:

En general, las operaciones asociadas a cualquier tipo serán aquellas cuyo resultado sea un elemento del mismo tipo. (1^a parte: Metodología de Programación)

Las operaciones asociadas serían +, -, *, % ó /, etc.

Real

Los números reales son todos aquellos números que tienen una parte entera y una parte decimal.

Su tamaño de almacenamiento puede variar según el subconjunto de números representado (4, 6 ó 10 bytes en el uso de flotantes, enteros largos, entre otros). Se representan por medio de la mantisa y un exponente

($1E-3 = 0'001$). Según Carmona Quintana en su libro *Resolución de problemas por computadora* se dice que “utilizando 24 bits para la mantisa (1 para el signo y 23 para el valor) y 8 bits para el exponente (1 para el signo y 7 para el valor)”.

El orden es de 10-39 hasta 10-38.

Por ejemplo

6.9, 33.00123, 3E-34.....

Las operaciones asociadas serían +, -, *, % o /, etc.

Lógico

Es un dato que puede tomar dos valores:

Cierto o Falso, guardando un '1' si es True o un '0' si es False.

Operaciones asociadas a este tipo de dato son: Todos los operadores lógicos y de relación.

Operadores lógicos

Mayor que >

Menor <

que

Igual ==

Diferente !=

Operadores de relación

Mayor que >

Menor <

que

Igual ==

Diferente !=

Y sus combinaciones

Mayor o igual
que

>=

Menor o igual
que

<=

“Cuando vamos a crear una condición podemos combinar tantos operadores como deseemos”

Carácter

Un carácter (letra) tienen un estándar de representación, el correspondiente en el código ASCII, que permite su identificación a nivel de manejo de memoria. Estos se muestran en la Tabla para los caracteres estándar (existe un código ASCII extendido que utiliza los 256 valores, es decir identifica mas caracteres). (García de Jalón J., Ignacio Rodríguez J. et all, Universidad de Navarra, Curso completo de visual Basic 6.0)

Sabiendo la fila y la columna en la que está un determinado carácter puede componerse el número correspondiente.

Los caracteres se representan entre comillas simples ‘a’, ‘D’... y abarcan:

- Números de ‘0’ ... ‘9’
- Letras del abecedario en minúsculas ‘a’ ... ‘z’
- Letras del abecedario en mayúsculas ‘A’ ... ‘Z’
- Caracteres especiales ‘!’, ‘#’, ‘(’ , Etc.

Introducción a la programación con C

Un espacio en blanco ‘ ’ también es un carácter por tratarse de un tipo de dato enumerado y con una representación como la de un entero.

Cadena

Los datos de este tipo contendrán una serie finita de caracteres. Podrán representarse de estas dos formas:
‘H’ ‘O’ ‘L’ ‘A’ == “HOLA”

Es diferente tener “H” indicando con las dobles comillas que es una cadena en lugar de tener ‘H’ indicando un carácter.

Las operaciones relacionadas con las cadenas son:
Concatenar(C1,C2), Longitud(C1), Extraer (C1,N)

Ejemplos:

```
C1 = "Hola", C2 = "Pepeeerl"  
C3 = Concatenar(C1,C2) //C3="HolaPepeeerl"  
X = Longitud(C2) //X=8  
C3=Extraer(C1,3) C3="Hol"
```

Los tipos de datos Entero, Real, Carácter, Cadena y Lógico son tipos predefinidos en toda la programación, o como también son llamados, datos primitivos. Por otro lado, muchos de los lenguajes de programación permite al usuario poder definir sus propios tipos de datos.

Tipos de variables básicas

En C tenemos los siguientes tipos de datos básicos: char, int, float y double. Los dos primeros son enteros y los otros dos del tipo real (admiten decimales). Estos cuatro tipos de datos se pueden convertir en algunos más cuando hacemos uso de los especificadores (o modificadores) en los tipos enteros. Al final, se nos quedan como aparecen en la tabla.

TIPO	BYTES	RANGO DE VALORES
Char	1	-128 a 127
Signed char	2	128 a 127
Unsigned char	1	-255 a 255
Short int	2	-32768 a 32767

Introducción a la programación con C

Signed short	2	-32768 a 32767
Unsigned short	2	0 a 65535
Long int	4	-2147483648 a 2147483647
Signed long	4	-2147483648 a 2147483647
Unsigned long	4	0 a 4294967295
Int (16 bits)	2	-32768 a 32767
Signed int (16bits)	2	-32768 a 32767
Unsigned int (16 bits)	2	0 a 65535
Int (32 bits)	4	-2147483648 a 2147483647
Signed int (32 bits)	4	-2147483648 a 2147483647
Unsigned int (32 bits)	4	0 a 4294967295
Float	4	- 3.402823466 E + 38 a3 +3.402823466 E + 38
Double	8	-1.7976931348623158E + 308 1.7976931348623158E + 308
Char	1	-128 a 127
Signed char	2	128 a 127

Como se puede observar en el cuadro anterior, los tipos enteros sin modificador de signo corresponden a tipos con signo (**signed**).

Algo parecido puede ocurrirnos con los **int**, que en los sistemas de 16 bits serán considerados como short y en los de 32 como **long**.

Si vamos a declarar una variable int que pueda salirse del rango de los shorts sería recomendable usar long (o long int) para que funcionara en cualquier sistema.

Todo esto para dar la portabilidad y confiabilidad al manejo de datos.

Constantes

Una constante es un dato en el que su valor fijo dentro de un programa. El término constante designa un valor específico y determinado, que se define al hacer un programa y que no cambia a lo largo del mismo. También a las constantes se les debe de colocar un nombre y deben ser de un determinado tipo de dato.

Existen dos tipos de constantes en C, estos son los siguientes:

Numéricas

Son aquellas con las cuales se pueden realizar operaciones de tipo aritmético.

Enteras

Una constante entera es una sucesión de dígitos precedidos o no del signo (+) O (-).

Ejemplo de ello tenemos:

10	0
2	-1
12350	-13500

Note como las constantes enteras pueden ser positivas o negativas. Inclusive, es posible escribir una constante entera así: +12

De igual forma se puede observar cómo no se usan separadores para los miles. El número 12350 se escribió simplemente así: 12350

Es bien importante tener en cuenta que no hace falta hacer declaraciones previas de las constantes enteras. Simplemente se escriben donde se necesitan.

Además, cabe notar que no se requieren símbolos especiales alrededor de las constantes enteras, es decir se escriben tal como son

Octales

Para las constantes enteras-octales se debe anteponer el número cero (0).

Ejemplo: 062, 035, 045.

Hexadecimales

Para las constantes enteras hexadecimales se debe anteponer el cero y la equis (0X).

Ejemplo

0x10	0x1A
0x2F4	0X20B

Introducción a la programación con C

Observe como las letras que representan dígitos del sistema hexadecimal, al igual que la equis que acompaña al cero, pueden ser escritas tanto en mayúscula como en minúscula. Para el computador es indiferente.

Decimal

Las constantes enteras-decimales se deben presentar tal cual son.

Ejemplo:

56,333 67,333123 78,0000 34.6565

Reales

Una constante de tipo real equivale al valor de un número real usual, es decir, una cantidad formada por una parte entera y una fraccionaria.

Los datos de tipo Real son llamados también números de punto flotante

Las constantes de tipo flotante también se escriben de manera similar a la forma como las empleamos en aritmética convencional.

Ejemplo:

12.5	1.6
-2341.189	-0.146987

Por supuesto, el principal indicador que se trata en las constantes de punto flotante es el punto.

En las constantes de punto flotante, No importa que su parte fraccionaria sea cero, la sola presencia del punto la convierte en una constante de punto flotante.

Note como las constantes de punto flotante también pueden ser negativas o positivas y que se puede usar el signo más (+) explícitamente.

El número de dígitos decimales no está limitado y puede ser cualquiera. Sin embargo, si un número constante de punto flotante tiene demasiados dígitos (entre enteros y decimales) entonces el computador terminará haciendo una aproximación para efectos de su almacenamiento dentro de la memoria del computador.

Los números constantes de punto flotante también pueden ser escritos usando la notación científica exponencial. Ejemplos de constantes de punto flotante escritas de esta manera son:

Introducción a la programación con C

6.2e13
-.36E-15
-83.423e+12
1E3

El significado numérico de cada una de estas constantes en forma respectiva es:

6.2x1013	o 620000000000000
-0.36x10-15	o -0.0000000000000036
-83.426x1012	o -83423000000000
1x103	o 1000

En C están definidas dos clases de constantes reales que son:

Float y double

Estas se definen normalmente en dígitos de precisión. Las magnitudes de estos dos tipos de reales dependen del método utilizado para representar los números en coma flotante.

Ejemplo:

float 64 Aproximadamente seis (6) dígitos de precisión.
double Aproximadamente doce (12) dígitos de precisión.

Alfanuméricas

Son conjuntos de caracteres con los cuales no se pueden realizar operaciones aritméticas.

Simbólicas

Estas Constantes son definidas en C para ampliar el ambiente de programación en el lenguaje, para hacer óptima una aplicación determinada.

Para llevar a cabo este tipo de constantes se utiliza la instrucción #define de la siguiente manera:

```
#define cadena de caracteres
```

Nota: En este tipo de instrucciones no hay punto ni coma, además puede existir cualquier número de blancos entre el #define y la cadena de caracteres.

La cadena de caracteres hace referencia a dos partes que son: una variable que va a ser constante durante la ejecución del programa y un valor determinado que va a tomar esa variable.

Ejemplo: #define N 10

El ejemplo anterior dejará constante la variable **N** con un valor de **10**.

El signo numeral (**#**) es sumamente importante ya que hace referencia a la portabilidad de los programas.

Lógicas

Son las que provienen de realizar comparaciones lógicas. Se podría definir como un estado de 1 o 0.

Ejemplo:

Datos lógicos:	falso	verdadero
	Sí	No
	1.	0

Variables

Una variable es algo que puede cambiar su valor dentro de un mismo programa (durante su ejecución). En sistemas es el nombre que se le da a un trozo de la memoria principal del computador, de tal manera que podamos depositar datos en ella.

A las variables se les debe dar un nombre, este nombre tiene como restricción principal el iniciar siempre con una letra.

Ejemplo:

Si quiero llevar la cuenta de las personas que van ingresando a un cine, entonces requerimos de una variable a la cual le asignamos un valor.

Ejemplo:

Persona _ cine = 1 y lo habrán hecho 2, 3, 4 y así sucesivamente.

Debido a que los datos variables pueden tomar muchos valores durante la ejecución de un programa, no pueden tener un valor preciso como el de las constantes. En lugar de ello, debemos asignarles un nombre en forma similar a como lo hacemos en el álgebra.

Introducción a la programación con C

De esta manera una variable podría llamarse simplemente **X** o también podría llamarse **Y**

En realidad se acostumbra que los nombres de las variables digan un poco más sobre su razón de existencia. Por ejemplo, para calcular el promedio de las personas que ingresaron a cine, podríamos usar una variable como:

Promedio personas o P_CINE

En general, podemos usar cualquier nombre para las variables respetando unas reglas simples:

- Los nombres de las variables deben empezar con una letra (no importa si es minúscula o mayúscula) o con el símbolo de subrayado_.
- Los nombres de las variables no pueden contener espacios, ni símbolos especiales como &, ., (, + y así sucesivamente. Tampoco pueden contener letras tildadas o eñes.
- Los nombres de las variables pueden contener dígitos y más símbolos de subrayado. El valor numérico de los dígitos dentro de los nombres de las variables no tendrá ningún significado particular para el computador, simplemente harán parte del nombre.
- No se pueden emplear como variables aquellas palabras que son consideradas propias del lenguaje y a las que se les llama **palabras reservadas**.

NOTA: “Una palabra reservada es una palabra usada por el lenguaje para algún fin particular y por tanto, está reservada para él y no puede ser usada como nombre de variable ni de ninguna otra clase de objetos.”

Lista de palabras reservadas

La lista de palabras reservadas por C, ANSI estándar son tan sólo 32, las cuales se listan a continuación:

Auto	break	case	char	const
Continue	default	do	double	Else
Enum	extern	float	for	Goto
If	int	long	register	return
Short	signed	sizeof	static	struct
Switch	typedef	union	unsigned	Void

Introducción a la programación con C

volatile while

Nota: Además de estas palabras se debe tener en cuenta no usar palabras de las funciones de las librerías usadas.

Se aconseja **elegir los nombres** de las variables de modo tal que permitan conocer qué papel representan dentro del programa.

Veamos entonces algunos ejemplos de nombres de variables válidos:

suma	resultado	calcuo	distancia_1	valor2
T	F	valorFinal	registroInicial	elemento01_02

El uso de las mayúsculas y minúsculas es de libre elección para el programador.

Se recomienda que los nombres de las variables no sean demasiado largos y que además sean fácilmente legibles.

Si es importante anotar que para el computador en lenguaje C, las mayúsculas y minúsculas se diferencian plenamente, de esta forma las variables:

total_factura, Total_factura, Total_Factura

Son tres variables claramente diferenciadas y válidas.

Ahora, veamos algunos ejemplos de nombres de variables no válidos junto con las razones por las que no lo son:

- **Total Operación** (no se puede usar espacios ni tampoco letras tildadas)
- **7ma** (no puede empezar con un dígito)
- **a + b** (no se deben emplear símbolos especiales como +)
- **Descuento.Porcentual** (no se puede emplear el punto .)

Para poder emplear una variable entera es necesario hacer una declaración previa de su existencia. Para ello se usa la palabra reservada **int**. Por ejemplo, para poder usar una variable entera **avance** dentro de nuestro programa se requerirá indicarlo así:

int avance;

Esta instrucción tiene un significado muy preciso: le estamos indicando al computador que dentro de nuestro programa usaremos una variable llamada **avance** que será de tipo entero.

Operadores

Los operadores nos permiten identificar la acción que se debe efectuar entre variables y/o constantes.

Un operador es un símbolo que indica al compilador que lleve a cabo ciertas manipulaciones matemáticas o lógicas.

En C hay tres clases generales de operadores: aritméticos, relacionales, lógicos y a nivel de bits, además C tiene operadores especiales para determinadas tareas.

Aritméticos

Estos tipos de operadores se pueden utilizar en todos los tipos predefinidos en C.

Dentro de estos están:

Suma(+)

Resta(-)

Multiplicación (*)

División (/)

Residuo de división (%) (resto de la división entera).

NOTA: El operador % se aplica solamente a constantes, variables o expresiones de tipo entero.

Ejemplo: $23 \% 4$ es 3, puesto que el resto de dividir 23 por 4 es 3.

En C el operador % de división en modulo proporciona el resto(residuo) de una división entera. Por ello, % no puede aplicarse a los tipos de coma flotante (reales)

Una **expresión** es una combinación de variables y constantes relacionadas mediante distintos operadores (1^a parte: Metodología de Programación). Un ejemplo de ello sería lo siguiente:

Función:

$$Y = (3 * x) - (x * x / 2)$$

Estos operadores pueden aplicarse a casi todos los tipos de datos predefinidos y permitidos en C.

```
int x, y, z;  
x = 30; y = 3;  
z = 10 %3  
x= (-b+raiz ((b*b) - (4*a*c))) / (2*a);
```

Introducción a la programación con C

Donde estrictamente hablando, sólo lo que está a la derecha del operador de asignación (=) es una expresión aritmética. En las expresiones se pueden introducir espacios en blanco entre operandos y operadores, por ejemplo: la expresión anterior se puede escribir también de la forma:

$$x = (-b + \sqrt{((b * b) - (4 * a * c)))} / (2 * a)$$

Los operadores, las constantes y las variables son los constituyentes de las **expresiones**.

Conversiones de tipos en las expresiones

Cuando en una expresión se mezclan constantes y variables de distintos tipos, se convierten a un tipo único. El compilador de C convierte todos los operandos al tipo del mayor operando, esto se hace operación a operación, tal como describen las siguientes reglas de conversión de tipos:

1.cualquier **char** y **short int** es convertido a **int**, cualquier **float** es convertido a **double**.

2.Para todos los pares de operandos, si uno de los operandos es **long double**, el otro operando se convierte a **long double**.

Si no, si un operando es **double**, entonces el otro se convierte a **double**.

Si no, si un operando es **long**, entonces el otro se convierte a **long**.

Si no, si un operando es **unsigned**, entonces el otro se convierte a **unsigned**.

Una vez aplicadas las reglas de conversión de tipos, cada par de operandos será del mismo tipo y el resultado de cada operación será del mismo tipo que el de los operandos.

Espaciado y paréntesis

Para dar mayor legibilidad, se pueden añadir tabulaciones y espacios a discreción en una expresión. Por ejemplo, las dos expresiones siguientes son la misma.

a) $X=10/y(127/x);$ b) $X = 10 / y (127 / x) ;$

El uso de paréntesis redundantes o adicionales no produce errores ni disminuye la velocidad de ejecución de una expresión. Conviene utilizar paréntesis para hacer más claro el orden en que se producen las evaluaciones, tanto para uno mismo como para los que tengan

Introducción a la programación con C

que seguir después el programa. (Tema 2: Componentes elementales de un lenguaje de programación)

Por ejemplo, si tuviera dos expresiones ¿cuál de las siguientes es más fácil de leer?

X = y/ 3-34 * Temp & 127;
X = (y/3) – (34*(Temp & 127));

Relacionales

Estos operadores hacen referencia a la relación existente entre unos valores y otros. Estos operadores son:

Operador	Acción
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
==	Igual
!=	Diferente

Nota “En el término operador relacional la palabra relacional se refiere a la relación entre unos valores y otros”

Asignación

Los **operadores de asignación** se relacionan con una variable, es decir, se asignan a la zona de memoria de una variable un valor proveniente de un cierto análisis (una variable es un caso particular de una expresión).

El operador de asignación más utilizado es el **operador de igualdad** (=). Primero se evalúa **expresión** y el resultado se pone en **nombre_variable**.

Otra forma de verlo es que todo aquello que se encuentre de lado izquierdo (nombre_variable) almacena el valor del lado derecho (expresión). Su forma general es:

nombre_variable = expresión;

Introducción a la programación con C

Este operador no debe ser confundido con la igualdad lógica (==), mayormente usado en la evaluación de condiciones.

Nota “el operador de asignación (=) representa una sustitución”, ya que toma el valor de la expresión y lo guarda en la variable nombre_variable.

Abreviatura

Estos operadores fueron creados con el propósito de simplificar y disminuir errores en los programas.

El lenguaje C admite abreviaturas que simplifican la escritura de ciertos tipos de sentencias de asignación, algunas de estas son:

+ = - = * = / = % ==

Ejemplo:

x = x + 1 es igual a x += 1 es igual x++
y = y * y + 1 es igual a y *= y + 1
a = 5;

Cuando el computador se encuentra con el operador igual, lo que hace es “calcular” el resultado total de todo lo que se encuentra a su derecha, toma este valor y lo almacena como contenido de la variable que se encuentra a la izquierda.

Usando otras palabras, el operador de asignación, **asigna** a la variable el resultado de la expresión. En el caso anterior a la variable a se le asigna el valor 5 (por que el resultado de calcular todo lo que se encuentra a la derecha del igual es simplemente el valor 5).

Lógicos

Los operadores lógicos son símbolos que permiten evaluar expresiones de manera falso o verdadero. Son utilizados para comprobar que se cumplen una o más condiciones.

Como **operadores lógicos** tenemos:

El operador Y (&&), el operador O (||) y el operador NO (!)

Sus formas generales son:

expresión1 **&&** expresión2,
expresión1 **||** expresión,
!(expresión)
&& And lógica bit a bit

Introducción a la programación con C

- ||** Or lógico bit a bit
- ^** Xor
- >>** Desplazamiento a la derecha
- <<** Desplazamiento a la Izquierda

Los operadores **&&** y **||** se pueden combinar entre paréntesis. Por ejemplo:

```
(4==1) || (-10== -10)           //el resultado es 1  
(2==2) && (3== -1)           //el resultado es 0  
((2==2) && (3==3)) || (4==0)    //el resultado es 1  
((6==6) || (8==0)) && ((5==5) && (3==2)) //el resultado es 0  
((250) && (330)) 250>0,330 > 0      //el resultado es 1
```

Es decir, se refieren a la forma en que los operadores relacionales pueden conectarse entre sí. Estos son:

Predicados		&& (Y)	(O)	! (NO)
p	q	P&&q	p q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

En el término operador lógico, la palabra lógico se refiere a las formas en que los operadores relacionales pueden conectarse entre sí, siguiendo las reglas de la lógica formal.

Sentencias

Al observar cualquier expresión, según Carmona Quintana en su libro *Resolución de problemas por computadora* se dice que “la variable que está a la izquierda del signo (**=**), el operador de asignación, la expresión aritmética y el carácter (**;**) constituyen una sentencia. Las sentencias son unidades completas ejecutables en sí mismas”.

Se verá que muchos tipos de sentencias incorporan expresiones aritméticas, lógicas o generales como componentes de dichas sentencias.

Instrucciones de entradas y salida

Salida	Entrada
printf	scanf

Introducción a la programación con C

Las librerías que se utilizan son:

#include <stdio.h> para funciones estándar de C ANSI

Formato de la instrucción **printf**

printf("Mensaje formato",nom_variable);

Cadenas y variables:

Se escribe la cadena, se separa la variable con una coma y después se escribe otra cadena. Se pone otra coma y se escribe la **cadena**.

Ejemplo:

printf("la base %d", b, "por la altura", a, "es la superficie", s)

Tipos de formatos

Manejo de datos numéricos

Identificador	Descripción
%d	Para el manejo de números enteros
%i	Para el manejo de números enteros
%f	Para el manejo de números reales
%o	Para el manejo de números en Octal
%x	Para el manejo de números en Hexadecimal
%p	Para el manejo de apuntadores

Manejo de caracteres y cadenas

Identificador	Descripción
%s	Para el manejo de cadenas
%c	Para el manejo de caracteres

Manejo de presentación por pantalla

Identificador	Descripción
%v	Para Visualizar el texto o números verticalmente
%h	Para Visualizar el texto o números horizontalmente

Ejemplo 1

```
int h;  
printf("Digite un numero");  
scanf("%d",&h)
```

Ejemplo 2

```
Int A=1;  
//Visualiza un mensaje y el contenido de la variable A:  
Printf(" el valor es %d ",A);
```

La instrucción de lectura Scanf ()

Toma los valores de forma interactiva desde el dispositivo de entrada estándar como lo es el teclado, introduciéndolos en las variables que forman parte de la instrucción. Los tipos de datos introducidos deben coincidir con los de las variables que los recogen. (Tema 2: Componentes elementales de un lenguaje de programación)

Hay dos formas de usarlas:

Formato scanf

```
scanf("tipo_formato", dirección Nom_Variable);  
scanf( "%d",      &     h    );
```

Donde la dirección me indica la posición donde queda almacenada la variable, en este caso la variable h.

Estructuras de programas Cualquier programa puede ser realizado con una combinación de las siguientes estructuras de sentencias que se definen a continuación:

- Secuencial**
- Selectiva**
- Repetitiva**

Estructura secuencial

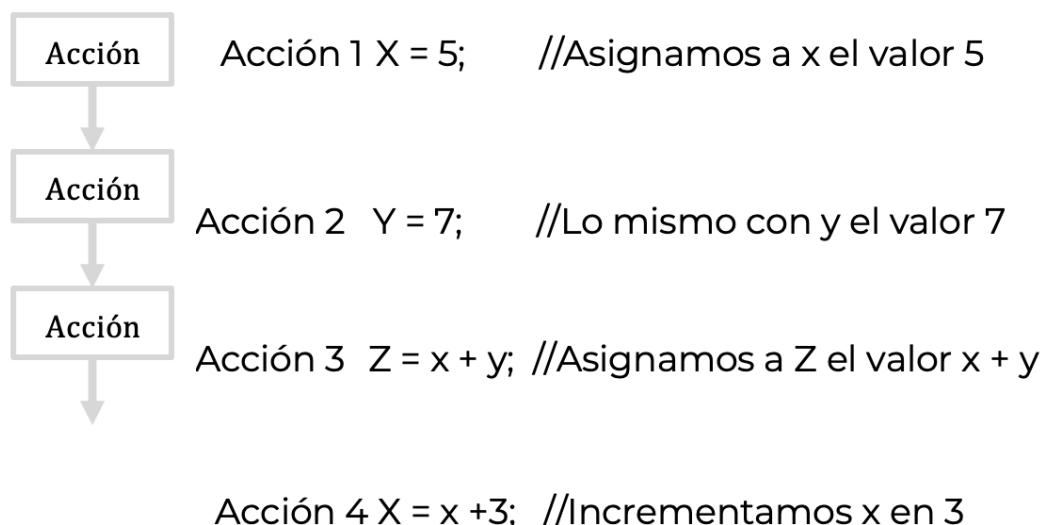
Aquí las instrucciones se ejecutan de acuerdo al orden en el cual se hayan escrito durante la creación del programa. Se ejecutan de forma secuencial (una tras otra) y el orden de ejecución permanece inalterado.

En primer lugar se ejecutaría la acción1, luego la acción2..., y así sucesivamente hasta llegar al fin del programa.

En el siguiente caso se evidencia una secuencia de acciones que consisten en la asignación de valores a variables y posteriormente incremento entre las mismas como se observa a continuación:

Diagrama de flujo

Pseudocódigo



Estructuras de control selectivas

En ciertos programas se pueden necesitar unas variaciones en el momento en que son ejecutados, esto es dependiendo si algunas condiciones son o no cumplidas. Las estructuras selectivas pueden tomar decisiones según ciertas condiciones, y según resulte su evaluación se procede a **ejecutar** o no una acción o acciones.

Las distintas estructuras de las que se disponen son:

if (si, entonces) y switch (según sea haga)

Introducción a la programación con C

La mayoría de las sentencias de control de programa de cualquier lenguaje de computadora, incluyendo a C, se basan en una prueba condicional que determina la acción que se ha de llevar a cabo.

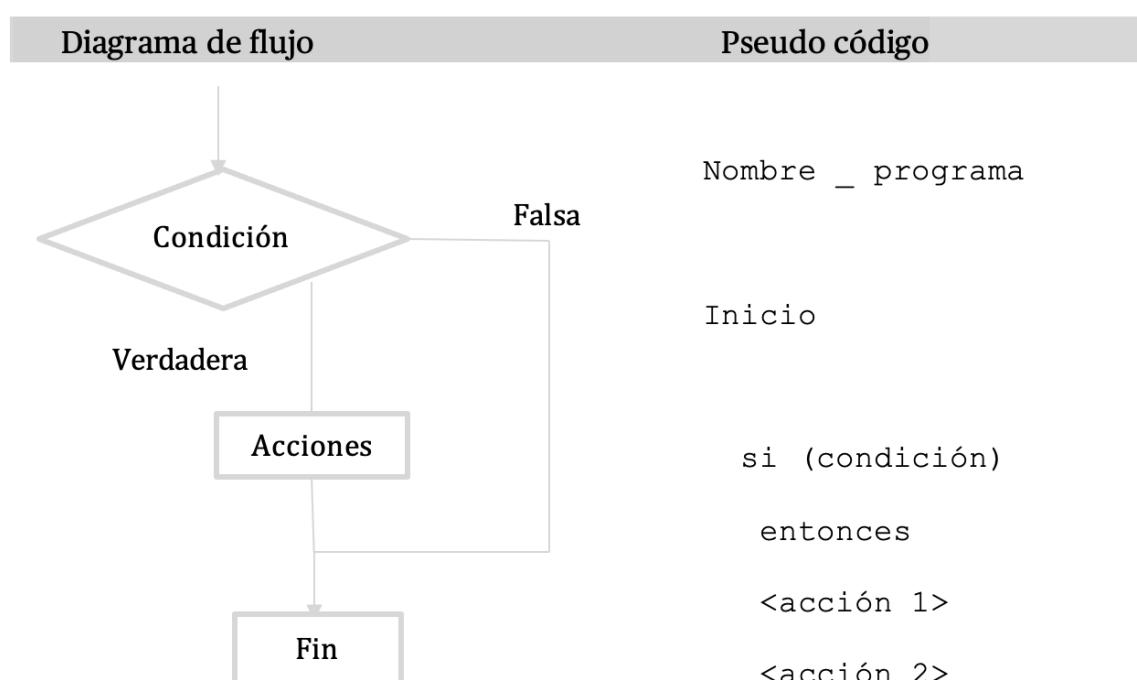
Según la sintaxis de C, una sentencia puede formarse como: una única sentencia, un bloque de sentencias o nada (en el caso de sentencias vacías). El término sentencia se usa para cualquiera de las tres posibilidades.

Selección Incompleta (simple)

Tomando como referencia el escrito *1^a parte: Metodología de Programación*, la estructura alternativa simple **si-entonces** ejecuta una o mas acciones cuando se evalúa como verdadero una determinada condición, es decir que dicha condición se cumple. Si la condición es falsa entonces el programa procede con la siguiente instrucción.

La representación gráfica de la estructura condicional simple se muestra en la siguiente figura:

Ejemplo 1



Lenguaje C

Introducción a la programación con C

1	#include <stdio.h> //librería
2	void main() //inició
3	{
4	if (expresión) //evalúa la condición
5	sentencia1; //accion1
6	sentencia2; //accion2
7	}

Ejemplo 2

El siguiente ejemplo ilustra una secuencia de condiciones que serán evaluadas a lo largo del programa y dependiendo de su veracidad serán ejecutadas las respectivas acciones.

Ejemplo

Un programa que indica el estado de un número. (Positivo, negativo o igual a cero)

Pseudocódigo

```
Inicio
{
    leer (numero);
    Si (numero = 0) entonces
        escribe ("el numero digitado es igual a cero");
    Si (numero<0) entonces
        escribe ("el numero digitado es negativo");
    Si (numero>0) entonces
        escribe ("el numero digitado es positivo");
}
```

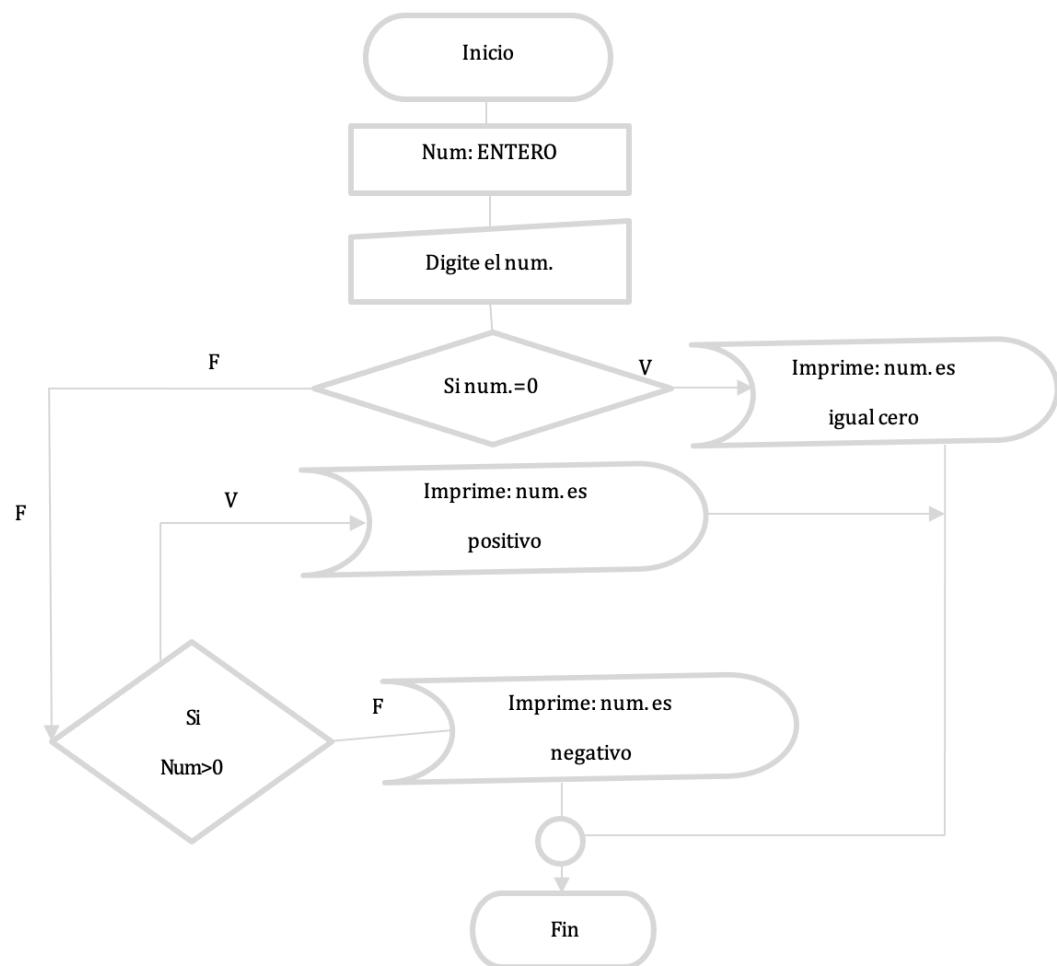
Pseudocódigo

```
inicio
{leer (x); // Obtenemos x desde el teclado
 si (x>=0) entonces
    y = raíz(x); // si x es positiva hallamos la raíz
    x = x * x; // hacemos x2
    escribe (x); // Escribimos el valor de x
}
```

Estructura de selección completa

Se realizarán una serie de instrucciones cuando se cumpla la condición y también se especifica qué acciones debe realizar si no se cumple la condición inicial.

Su pseudocódigo es el siguiente:



Pseudocódigo

(Con acción compuesta)

```
si (condición) entonces
{ <acción 1>};
sino
{ si (condición) entonces
{<acción 2>;}
sino
{<acción 3>;}
}
```

Ejemplo 2

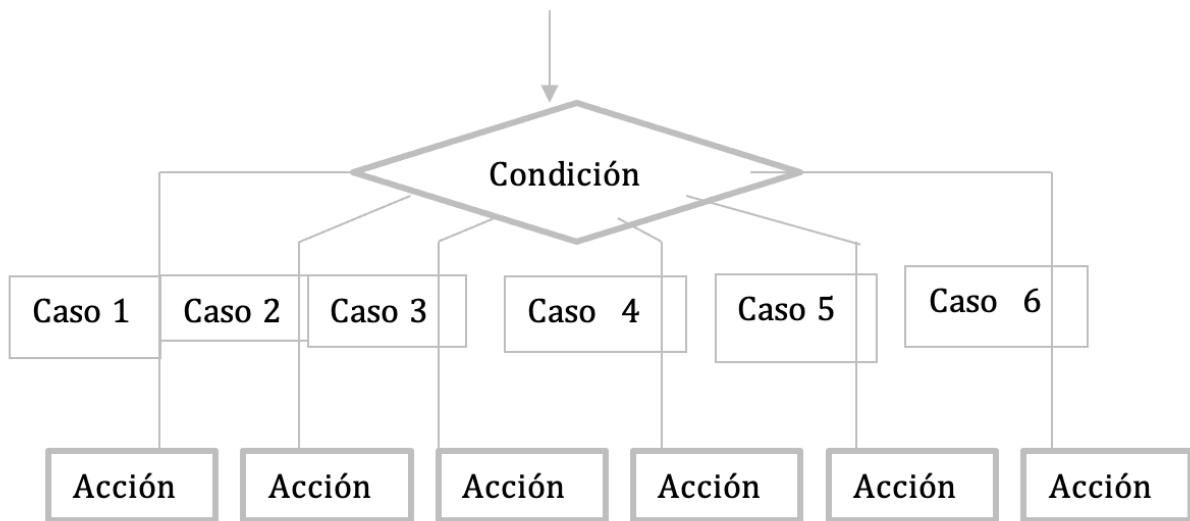
```
Inicio
{
    leer (edad);
    si (edad>=18) entonces
    {
        escribe ("usted es mayor de edad");
        escribe (edad);
    }
    sino
    {
        escribe ("usted es menor de edad")
        escribe (edad);
    }
}
```

Switch (según sea)

C incorpora una estructura de decisión múltiple llamada **switch**. Esta compara el valor de una variable con respecto a un conjunto de posibles opciones. Cuando se encuentra un valor que es igual al de la variable comparada el programa ejecuta una sentencia o bloque de sentencias.

Introducción a la programación con C

La forma general de la sentencia **switch** es:



```
switch (variable)
{
    case constante1: secuencia de sentencias
        break;
    case constante2: secuencia de sentencias
        break;
    case constante3: secuencia de sentencias
        break;
    default: secuencia de sentencias;
}
```

Estructuras de control repetitivas

Algunas veces se requieren realizar tareas que exigen repetir un determinado paso un cierto número de veces. Este tipo de estructuras son de gran importancia y utilidad puesto que permiten ahorrar muchas líneas de código, pero para su uso hay que tener en cuenta ciertos aspectos que involucran su funcionalidad:

- a. El conjunto de tareas debe ser finito, es decir las acciones que hacen parte del ciclo deben alguna vez llegar a un límite de operación.
- b. La cantidad de veces que se repita dicho conjunto de instrucciones también debe ser finito, esto indica que el **incrementador/decrementador** para dicho ciclo sumará o restará

Introducción a la programación con C

cierta cantidad de veces dependiendo de su condición, que a su vez puede ser explícita o implícita:

- Una condición es **explícita** cuando depende únicamente de la misma ejecución del programa.
 - Una condición es **implícita** cuando depende únicamente de la voluntad del usuario y por lo tanto la cantidad de iteraciones o repeticiones del ciclo podrá llegar a ser diferente cada vez, dependiendo del gusto de cada usuario.
- c. Igual que en las demás estructuras, los ciclos también deben estar demarcados por un inicio y un fin.

Por ejemplo: se desean sumar 35 números ingresados por teclado, por ejemplo, las edades de los estudiantes en un salón de clases.

Con lo visto hasta ahora, algo que se podría hacer es leer y asignar en variables de manera independiente a cada número de los 35 propuestos. Una variable almacena un número, mientras que la variable SUMA va calculando las sumas parciales, es decir, va sumando cada número con el siguiente en ingresar. La variable SUMA se hace igual a cero y a continuación se incrementa en el valor del número cada vez que uno de ellos se lea.

Pseudocódigo:

Inicio

```
entero número, suma;  
suma = 0;  
Leer (numero);  
suma = suma + numero;  
Leer (numero);  
suma = suma + numero;
```

...

Fin

Lo anterior según el funcionamiento planteado se repetiría otras 33 veces hasta obtener el total de números, en otras palabras el algoritmo repite muchas veces las mismas instrucciones.

Si se desea usar un bucle, un proceso que se repite cuantas veces se indique, y aplicarlo a la suma del conjunto de números, se debe conocer el total de elementos a manejar.

En el ejemplo anterior se repetiría **desde 1 hasta 35** la acción de pedir la edad.

Estructura mientras

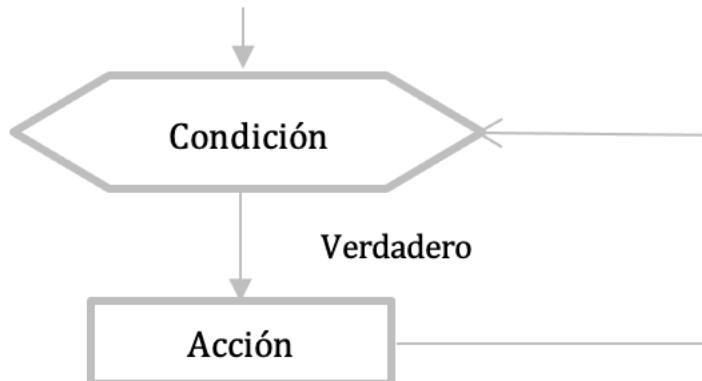
Aquí las instrucciones o grupo de instrucciones se repetirán **mientras** se cumpla una condición y en el momento en que no se cumple, el bucle se detiene:

Una variable servirá como contador, es decir que contendrá el número de veces que se quiere repetir la secuencia de código, por lo tanto debemos **conocer obligatoriamente** el número de veces (iteraciones) que se realizaran.

Pseudocódigo:

```
i = índice  
Mientras i = valor_inicial hasta i = valor_final  
<acción>;  
incremento valor
```

Diagrama de flujo del Mientras



Ejemplo 1: el siguiente programa determina la cantidad de números impares comprendidos entre 1 y 100. Veamos:

Introducción a la programación con C

```
Inicio i = 1; h = 100;  
mientras (i) hasta  
(h) inicio  
    escribe ("cantidad de números impares es:");  
    escribe (i);  
  
    i = i + 2;  
fin mientras  
Fin
```

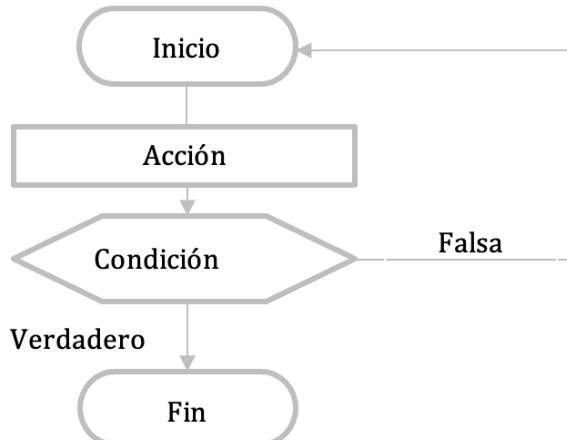
El formato del bucle **mientras** de C se encuentra de una forma o de otra en todos los lenguajes de programación de procedimiento. Este bucle también se destaca por proporcionar en C una sorprendente potencia y flexibilidad.

Por otra parte, si mostramos una forma distinta de ejecutar un bucle, tendríamos la sentencia **for** que es la siguiente:

For (inicialización; condición; incremento)
 Sentencia;

Estructura haga mientras

Esta estructura **se ejecutará siempre al menos una vez**. Cuando una instrucción **haga mientras** se ejecutan las acciones o instrucciones y a continuación se evalúa la condición. Si se evalúa como falsa el cuerpo del programa se repite, caso contrario si es *verdadera*, donde el ciclo termina y el programa pasa a ejecutar la siguiente instrucción.



Ejemplo 1: lee un número entero y arroja el valor de su sumatoria:

```
Inicio a = 0; b  
= 0; leer  
(numero);  
haga  
    a = a + 1;  
    c=a+b;  
    b=c;  
mientras (a<= numero)  
escribe ("la sumatoria es:  
"); escribe (c); fin
```

Ejemplo 2: un programa que lee un número entero y determina cuantos dígitos lo componen. Veamos:

```
Inicio  
nd = 0; leer(a);  
haga  
    a = a/10;  
    nd = nd + 1;  
mientras (a>=1)  
escribe ("el numero tiene: ");  
escribe ("%d dígitos", nd);  
Fin
```

Ejercicios en C

1. Realizar un programa que determine si un número es par o no

```
1 #include "stdio.h"
2 #include "conio.h"
3 int i,numero;
4 void main (void)
5 {
6     for(i=1;i<=10;i++)
7     {
8         printf("\n\n digite el numero %d:",i);
9         scanf("%d",&numero);
10    if((numero%2)==0)
11        printf(" el número es par:%d",numero);
12    else
13        if((numero%2)!=0)
14            printf(" el número es inpar :%d ",numero);
15    }
16 }
```

2. Realizar un programa que realice la suma de pares e impares de 10 números ingresados por teclado

```
1 #include <stdio.h>
2 #include <conio.h>
3 int i, suma, suma1, num;
4 void main(void)
5 {
6     for(i=1;i<=10;i++)
7     {
8         printf("digite el numero: ");
9         scanf("%d",&num);
```

Introducción a la programación con C

```
10         if((num%2)==0)
11             suma=suma+num;
12         else
13             if((num%2)!=0)
14                 sumal=sumal+num;
15     }
16     printf("\n numeros pares %d \t",suma);
17     printf("\n total impares %d\t",sumal);
18     getch();
19 }
```

3. Realizar un programa que lea un número y determine si es capicúa o no

TALLER DE CADENAS (Encriptación)

Problema 1

• Un simple (y fácil de descifrar) método de cifrado es el de escribir una palabra al revés (de atrás hacia delante). Por tanto la cadena: "Hola mi nombre es Pepa" sería cifrada por "aloH im erbmon se apeP". En este algoritmo la clave está implícita.

Problema 2

- El alfabeto cifrado es un alfabeto normal el cual está desplazado un número determinado de posiciones hacia la izquierda o la derecha. Por ejemplo, aquí el cifrado César está usando un desplazamiento de seis espacios hacia la derecha:

<ul style="list-style-type: none">• Texto original: ABCDEFGHIJKLMNOPÑOPQRSTUVWXYZ Texto codificado: GHIJKLMNOPÑOPQRSTUVWXYZABCDEF

- Para codificar un mensaje, simplemente se debe buscar cada letra de la línea del texto original y escribir la letra correspondiente en la línea codificada. Para decodificarlo se debe hacer lo contrario.

- Texto original: WIKIPEDIA, LA ENCICLOPEDIA LIBRE
Texto codificado: CÑPÑVKJÑG, QG KSIÑIQUVKJÑG QÑHXK

Problema 3

Bueno. Entonces agarremos cada uno de los símbolos de recién y convirtámoslo a números con su equivalente ASCII. Después de eso, sumamos dígito a dígito el mensaje y la clave.

+	H o l a M u n d o !
+	@ s e c p r e @ s e
<hr/>	
+	72 111 108 97 77 117 110 100 111 33
<hr/>	64 115 101 99 112 114 101 64 115 101
<hr/>	136 226 209 196 189 231 211 164 226 134

Teniendo esta suma, ya estaría la frase cifrada, pero se observa la tabla ASCII, éstos números pueden caer en caracteres que pueden no tener representación en el teclado, o mejor dicho, en la escritura tradicional (por ej. si alguno cae en DEL o algo así, y se quiere escribir el mensaje cifrado en una web, o mandar por sms)

Capítulo 3

Capítulo 3: Arreglos y métodos de ordenamiento

En este capítulo se dará una introducción a los arreglos más básicos: los vectores y las matrices.

Se verán primero los vectores, introducción de valores y manejo de estos. Después se pasará a ver métodos de ordenamiento para los vectores, métodos que ayudan a organizar un vector de modo que sus valores vayan de menor a mayor.

Después se finalizará con una lección acerca de las matrices, el ingreso de datos y la manipulación de estos datos dentro de la matriz.

Arreglos y métodos de ordenamiento

Estructuras de agrupamiento de variables

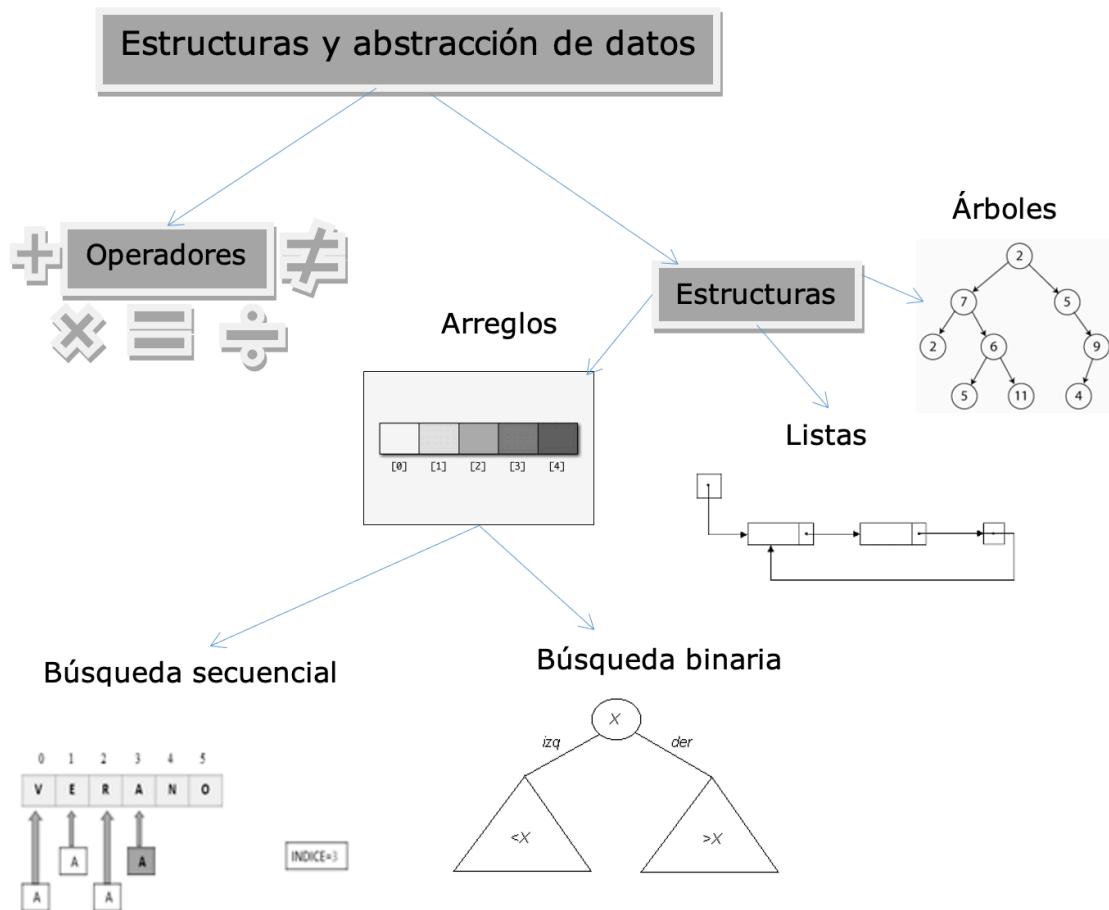


Figura 1.0.2. Mapa Mental 2

Arreglo

Es una estructura formada por varios elementos de la misma naturaleza, también podemos decir que un arreglo es una colección finita, homogénea y ordenada de elementos. Esta debe ser:

- **Finita:** Porque tiene un final.
- **Homogénea:** Porque todos los elementos del arreglo tienen el mismo tipo de dato:
Entero, reales, cadena de caracteres, etc.
- **Ordenada:** Porque se puede determinar cuál es el orden específico de cada elemento del arreglo a través del índice

Arreglo (vectores):

En C un arreglo es una lista de elementos del mismo tipo, que se referencia por un nombre común.

Los arreglos constituyen un modo adecuado de trabajar grupos de datos relacionados y de gran tamaño.

Para declarar un arreglo unidimensional, se utiliza la forma general:

Tipo de dato nombre_de_variable [tamaño]

Por ejemplo, para declarar un arreglo entero de 7 elementos llamado array:

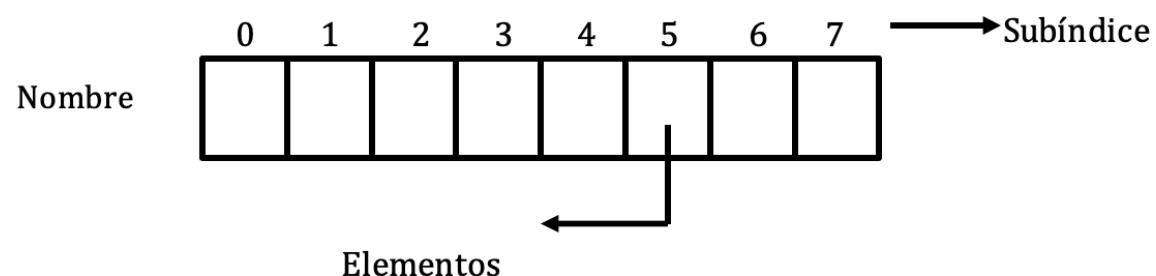
Int array[8];

Índice:	0	1	2	3	4	5	6	7
Contenido:	7	3	6	9	8	4	0	2

Un vector está compuesto por 3 componentes:

- Por el nombre.
- Por un Subíndice.
- Por contenido (los elementos)

Representación Gráfica de un arreglo



Declaración

Tipo de dato nombre_variable [cantidad de elementos];

Por ejemplo:

```
int var[10];           //Para declarar enteros  
char nombre[50];       //Para declarar cadenas  
float numeros[200];    //Para declarar decimales
```

```
long double cantidades[25]; //Para declarar cantidades grandes
```

En el primer caso se está declarando un vector de 10 variables enteras, cada una de ellas se manejará de forma individual por medio del subíndice que sigue al nombre del mismo, es decir:

var1 [0], var1 [1], etc., hasta var1 [9].

Nótese que la cantidad de elementos es 10, pero su numeración va de **0 a 9** y no de 1 a 10. En cada uno de estos elementos podemos almacenar un elemento del mismo tipo. En resumen un vector de N elementos tiene subíndices válidos entre **0 y N - 1**, cualquier otro número usado y que esté fuera de este límite traerá datos provenientes de otras zonas de memoria, lo que en cuyo caso generará un error al momento de la ejecución del programa en cuestión.

Recordando el subíndice, este permite referenciar a cada elemento del arreglo de forma individual, por ejemplo:

```
var1[5]=40;  
contador = var1[3] + 7;  
if (var1 [0] >= 37)
```

También es posible utilizar como subíndice expresiones aritméticas, valores enteros retornados por funciones, etc, así se podrá escribir:

```
printf(" %d ", var1[ ++i ]);  
var1 [8] = var1[ i + j ];
```

La inicialización de un vector local puede realizarse en su declaración, dando una lista de valores iniciales:

```
int numero[8] = { 4 , 7 , 0 , 0 , 0 , 9 , 8 , 7 };
```

Introducción a la programación con C

Obsérvese que la lista está delimitada por llaves. Otra posibilidad sólo válida cuando se inicializan todos los elementos del vector es escribir:

```
int numero[ ] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Se omite la declaración de la cantidad de elementos dentro de corchetes [] ya que este se indica de manera implícita a través de la lista de valores constantes (los valores entre llaves {...}).

También se puede inicializar parcialmente un vector, por ejemplo:

```
int numero[10] = {1, 1, 1} ;
```

En este caso los tres primeros elementos del vector valdrán 1 (**uno**) y las posiciones restantes toman el valor de **cero** en caso de que la declaración sea global o local.

Ejemplo:

```
int M[5], I;  
for ( I=0; I<5; I++) {  
    M[ I ]=I;  
}
```

M	0	1	2	3	4	Subíndice
	0	1	2	3	4	Elementos

El ejemplo anterior va llenando el arreglo uno por uno a medida que el ciclo se va ejecutando.

M[0] =0;

Como imprimir un vector por pantalla:

```
int M[5], I;  
for ( I=0; I< 5; I++)  
{  
    printf ("los vectores son %d ", M[I]);  
}
```

Ejemplo 1:

Leer un vector de 10 elementos y almacenarlos en un vector. Determinar la suma sólo de los numero pares.

Introducción a la programación con C

```
1 #include <stdio.h> //Declaración de la librería del programa
2 #include <conio.h>
3 int v[10]; // Declaración del vector y las variables
4 int i, suma;
5 void main (void) // Inicio del programa
6 {
7     for(i=1; i<=10; i++) // Inicio de ciclo desde 1 hasta 10
8     {
9         printf ("\n digite el numero %d:",i);
10        //Entrada de datos por teclado
11        scanf ("%d", &v[i]); //Lee y almacena en el vector
12    }
13    for(i=1;i<=10;i++) //Inicia el ciclo para recorrer el vector
14    {
15        //Evalúa si los elementos del vector son pares o no:
16        if ( (v[i] %2) == 0)
17            //Si se cumple la condición se suma los datos:
18            suma = suma + v[i];
19        else //Si no cumple la condición escriba no es par
20        {
21            printf (" no es número par ");
22            //imprime el resultado de la suma:
23            printf ("\n la suma de numeros pares es:%d ",suma);
24        }
25    }
26    getch(); //Visualiza en pantalla.
27 } //final del programa.
```

Ejercicios

1. Realizar un programa que lea la temperatura del medio día de un mes y después informar la temperatura media mensual, así como los días más fríos y cálidos.

2. Llenar un vector de 10 elementos y determinar cuántos números son múltiplos de 3.

3. Llenar un vector de 10 elementos y determinar cuántos números negativos, positivos y ceros tiene el vector.

4. Llenar un vector de 10 elementos e invertir el vector.

5. Llenar un vector de 10 elementos, digitar un número y determinar cuántas veces se repite el número.

6. Borrar un elemento del vector.

7. Llenar un vector de 10 elementos y determinar cuántos números están en los siguientes rangos.

num > 0

num > 3 num < 6

num > 8

8. Calcular el promedio de 50 valores almacenados. Determinar además cuantos son mayores que el promedio, imprimir el promedio, el número de datos mayores que el promedio y una lista de valores mayores que el promedio.

9. Almacenar 500 números en un vector, elevar al cuadrado cada valor y almacenar el resultado en otro vector.

10. Imprimir el vector original y el vector resultante.

11. Almacenar 30 números en un vector, imprimir cuántos son ceros, cuántos son negativos y cuántos positivos. Imprimir además la suma de los negativos y la suma de los positivos.

12. El dueño de una cadena de tiendas de artículos deportivos desea controlar sus ventas por medio de una computadora. Los datos de entrada son:

- El número de la tienda (1 a 50)
- Un número que indica el deporte del artículo (1 a 20)
- El costo del artículo.

13. Hacer un programa que escriba al final del día lo siguiente

- Las ventas totales en el día para cada tienda
- Las ventas totales para cada uno de los deportes.
- Las ventas totales de todas las tiendas.

14. Se tienen 50 alumnos los cuales se tiene la información sobre las calificaciones de la materia de LENGUAJES ALGORITMICOS.

Diseñe un diagrama que imprima:

- Cantidad de alumnos que aprobaron la materia.
- Cantidad de alumnos que tienen derecho a nivelación.
- El (o los) numero (s) de control de lo(s) alumno(s) que haya (n) obtenido la máxima calificación final.

15. La multiplicación “a la rusa” permite realizar el producto de dos números enteros utilizando solo sumas y divisiones o

productos por dos. El método consiste en realizar sucesivamente las siguientes operaciones: multiplicar uno de los números por dos y obtener la división entera del otro, también por dos. Se detiene el proceso al obtener como cociente de la división un uno. Por ejemplo, para multiplicar 22 por 64 se obtienen las dos columnas siguientes:

22	64
11	128
5	256
2	512
1	1024
22 x 64 =	1408

Problema 1. Escribir un programa que ingrese datos hasta que se introduce un número negativo y calcule:

- 1. Llenar //procedimiento
- 2. Mostar //procedimiento
- 3. La media. // función
- 4. EL máximo. // función
- 5. EL mínimo. // función
- 6. Contar el número de primos //en el procedimiento incluir la función booleana que diga si es primo o no
- 7. Colocar los pares en una lista_1 y los impares en lista_2 // procedimiento

16. Luego se suman los números de la columna de la derecha que corresponden a números impares en la columna de la izquierda, esto es para el caso de ejemplo: $128+256+1.024=1.408$. El resultado de la suma es el producto de los números iniciales. Elaborar un programa que multiplique mediante este método dos números enteros introducidos por pantalla.

- 17. Leer n números enteros por teclado, adicionalmente leer dos números. Determinar si los dos números forman parte de la secuencia de números inicial leídas. La secuencia de números finaliza en -1.
- 18. Conocida una secuencia de números enteros positivos finalizada en -1 (fin de secuencia), desarrolle un programa que determine:
 - a. ¿Cuántos de esos números son pares?
 - b. ¿Cuál es el valor del número máximo?
 - c. ¿Cuál es el valor del número mínimo?

19. Problema 2. Realizar las siguientes funciones o procedimientos con un menú (estructura case).....

- Potencia (base exp);
- Valor_Absoluto (a);
- Cuadrado (x);
- Potencia (Valor_Absoluto (a) , exp);
- Suma (Potencia(base, exp), Cuadrado(x)) ;
- Menu(.....);

//ciclo do {
 switch (op)

}while()

Algoritmos de Ordenamiento

Existen dos tipos de métodos de ordenación:

- Ordenación interna
- Ordenación externa

Ordenación interna

Generalmente los métodos de ordenación interna se aplican a los arreglos unidimensionales (vectores), ya depende del número de elementos en este último el que determinará la eficiencia del método . En este caso aplica para datos de menos de 100 elementos.

Los principales algoritmos de ordenación interna son:

Método de Selección:

Este método consiste en buscar el elemento más pequeño del arreglo y ponerlo en primera posición, de esa manera continua con el resto de elementos hasta ya haber recorrido y seleccionado cada uno.

40	21	4	9	10	35
----	----	---	---	----	----

Los pasos por seguir son:

Paso 1

Se busca el elemento más pequeño y la posición en que se encuentra.

Introducción a la programación con C

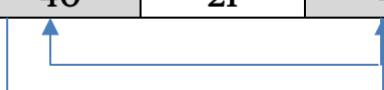
0 1 2 3 4 5 INDICE

40	21	4	9	10	35	
----	----	---	---	----	----	--

En este caso es el número 4 y se encuentra en la posición 2 del vector. Se intercambia el 4 por el 40 que se encuentra en la primera posición.

0 1 2 3 4 5 INDICE

40	21	4	9	10	35	
----	----	---	---	----	----	--



El vector se queda así:

0 1 2 3 4 5 INDICE

4	21	40	9	10	35	
---	----	----	---	----	----	--

Paso 2

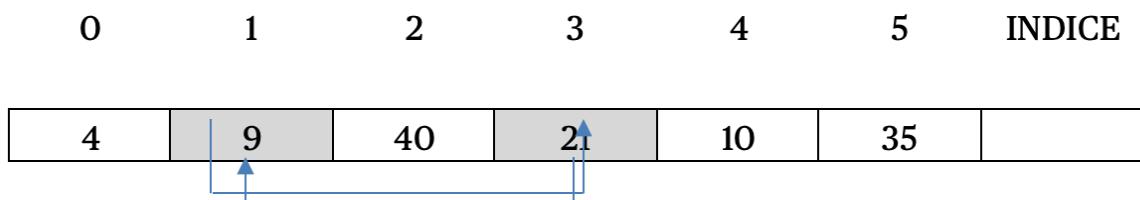
Se busca el segundo elemento más pequeño y la posición en que se encuentra, en este caso es el número 9.

0 1 2 3 4 5 INDICE

4	21	40	9	10	35	
---	----	----	---	----	----	--

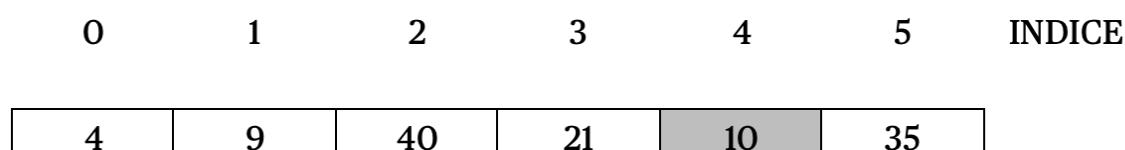
Introducción a la programación con C

Se incrementa una posición el índice en este caso pasa de posición cero al uno cambia el 9 por el 21.

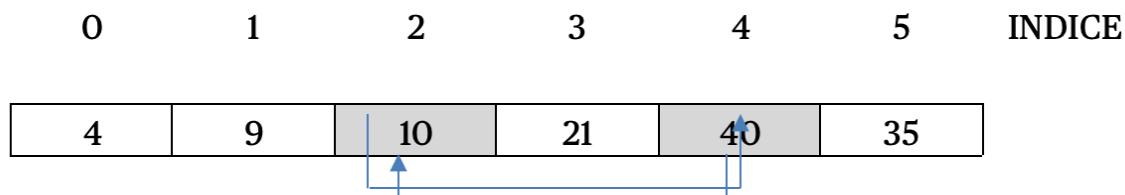


Paso 3

Se busca el tercer elemento más pequeño y la posición en que se encuentra, en este caso es el número 10.

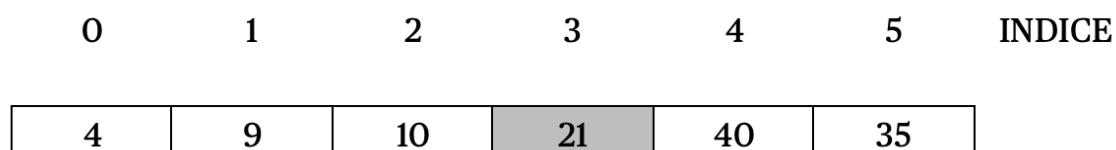


Se incrementa una posición el índice, en este caso pasa de posición uno al dos cambia el 10 por el 40.



Paso 4

Se busca el cuarto elemento más pequeño y la posición en que se encuentra, en este caso es el número 21.



Introducción a la programación con C

Se incrementa una posición el índice, en este caso pasa de posición dos al tres. Se compara si el número que se encuentra en la tercera posición es menor que el número menor buscado, vemos que es igual, por lo que no lo movemos de posición y se pasa de nuevo a buscar el quinto elemento más pequeño, se ve que es 35 y su posición es 5, ahora se incrementa el índice en uno y pasa a ser 4, entonces se pueden intercambiar:

0	1	2	3	4	5	INDICE
4	9	10	21	40	35	

0	1	2	3	4	5	INDICE
4	9	10	21	35	40	



Un arreglo tiene N elementos, el número de comprobaciones que hay que hacer es de orden de:

N*(N-1)/2, luego el tiempo de ejecución está en **O (n²)**

Algoritmo en c (Método de selección):

1	int arreglo[N];
2	int i, j, menor, aux;
3	for(i =0; i <N-1; i++)
4	{
5	for(j =i+1, menor=i; j<N; j++)
6	if (arreglo[j] < arreglo[menor])
7	menor = j; // el menor pasa a ser el elemento j.
8	aux = arreglo[i]; // Se intercambian los elementos
9	arreglo[i] = arreglo[menor];
10	arreglo[menor] = aux;

11 }

Método de Burbuja:

Consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que estén todos ordenados.

Con el arreglo anterior:

40	21	4	9	10	35
----	----	---	---	----	----

Primera pasada:

- Se cambia el 21 por el 40.

21	40	4	9	10	35
----	----	---	---	----	----

- Se cambia el 40 por el 4.

21	4	40	9	10	35
----	---	----	---	----	----

- Se cambia el 9 por el 40.

21	4	9	40	10	35
----	---	---	----	----	----

- Se cambia el 40 por el 10.

21	4	9	10	40	35
----	---	---	----	----	----

- Se cambia el 35 por el 40.

21	4	9	10	35	40
----	---	---	----	----	----

Segunda pasada:

Vector inicial.

21	4	9	10	35	40
----	---	---	----	----	----

- Se cambia el 21 por 4.

4	21	9	10	35	40
---	----	---	----	----	----

- Se cambia el 9 por el 21.

4	9	21	10	35	40
---	---	----	----	----	----

- Se cambia el 21 por el 10

4	9	10	21	35	40
---	---	----	----	----	----

- ¿Se cambia 21 por 35?
- No intercambia 21 por el 35, ya que el 35 es mayor que el 21, solo intercambia en sentido contrario.

4	9	10	21	35	40
---	---	----	----	----	----

- No intercambia 35 por el 40, ya que el 40 es mayor que el 35, solo intercambia en sentido contrario.

4	9	10	21	35	40
---	---	----	----	----	----

Introducción a la programación con C

Tercera pasada:

- No intercambia 9 por 4 pero va incrementando el índice

4	9	10	21	35	40
---	---	----	----	----	----

- No intercambia 10 por 9

4	9	10	35	21	40
---	---	----	----	----	----

- No intercambia 35 por 10

4	9	10	35	21	40
---	---	----	----	----	----

- Intercambia 21 por 35

4	9	10	21	35	40
---	---	----	----	----	----

Al terminar, los elementos están ordenados, pero para comprobarlo se tendrá que acabar esta tercera comprobación y hacer una cuarta.

Si el arreglo tiene N elementos, para estar seguro de que el arreglo está ordenado, hay que hacer N-1 pasadas, por lo que habría que hacer **(N-1)*(N-i-1)** comparaciones, para cada **i desde 1 hasta N-1**. El número de comparaciones es, por tanto, **N(N-1)/2**, lo que nos deja un tiempo de ejecución, al igual que en la selección, en **O (n²)**.

Algoritmo en c:

1	int arreglo[N];
2	int i, j, aux;
3	for (i=0; i<N-1; i++)
4	for(j=0; j<N-i-1; j++) //Mirar los N-i-1 pares
5	if(arreglo[j+1] < arreglo[j])
6	{

Introducción a la programación con C

7	aux=arreglo[j+1];
8	arreglo[j+1] = arreglo[j];
9	arreglo[j] = aux;
10	}

Método de Inserción Directa:

En este método lo que se hace es tener una sub lista ordenada de elementos del arreglo e ir insertando el resto en el lugar adecuado para que la sub lista no pierda el orden. La sub lista ordenada se va haciendo cada vez mayor, de modo que al final la lista entera queda ordenada. (Facultad de Ciencias de la computación, Benemérita universidad Autónoma de Puebla .Algoritmos de ordenación)

Ejemplo: se tiene

0 1 2 3 4 5 Indice

40	21	4	9	10	35
----	----	---	---	----	----

La primera sub lista ordenada es 40.

40	21	4	9	10	35
----	----	---	---	----	----

Se inserta el 21

40	21	4	9	10	35
----	----	---	---	----	----

{40,21,4,9,10,35} <- {40,40,4,9,10,35} <- aux = 21;

Ahora la sub lista ordenada es {21,40}.

21	40	4	9	10	35
----	----	---	---	----	----

<-- Insertamos el 4

21	40	4	9	10	35
----	----	---	---	----	----

{ 21,40,40,9,10,35 } <-- aux = 4;

{ 21,21,40,9,10,35 } <-- aux = 4;

Ahora la sub lista ordenada es {4, 21, 40}.

4	21	40	9	10	35
---	----	----	---	----	----

Introducción a la programación con C

<-- Insertamos el 9:

4	21	40	9	10	35
---	----	----	---	----	----

{4,21,40,40,10,35} <- aux = 9;

{4,21,21,40,10,35} <- aux = 9;

Ahora la sub lista ordenada es {4, 9, 21, 40}.

4	9	21	40	10	35
---	---	----	----	----	----

<--Se inserta el 10:

4	9	21	40	10	35
---	---	----	----	----	----

{4,9,21,40,40,35} <- aux = 10;

{4,9,21,21,40,35} <- aux = 10;

Ahora la sub lista ordenada es {4, 9, 10, 21, 40}

Y por último se inserta el 35

4	9	10	21	40	35
---	---	----	----	----	----

{4, 9, 10, 21, 40, 40} <- aux = 35;

4	9	10	21	35	40
---	---	----	----	----	----

El arreglo está ordenado

En el peor de los casos, el número de comparaciones que hay que realizar es de **N*(N+1)/2-1**, lo que nos deja un tiempo de ejecución en **O (n²)**. En el mejor caso (cuando la lista ya estaba ordenada), el número de comparaciones es N-2. Todas ellas son falsas, con lo que no se produce ningún intercambio, por lo tanto el tiempo de ejecución está en O(n).

El caso medio dependerá de cómo están inicialmente distribuidos los elementos. Se observa que cuanto más ordenada esté inicialmente más se acerca a **O(n)** y cuanto más desordenada, más se acerca a **O(n²)**.

El peor caso es igual que en los métodos de burbuja y selección, pero el mejor caso es lineal, algo que no ocurría en éstos, con lo que

Introducción a la programación con C

para ciertas entradas se puede tener ahorros en tiempo de ejecución.

Algoritmo en c:

1	int arreglo[N];
2	int i, j, aux;
3	for(i=1; i<N; i++)
4	{
5	aux=arreglo[i];
6	for(j=i-1;j>=0;j--)
7	{
8	if(aux>arreglo[j])
9	{
10	arreglo[j+1]=aux;
11	break;
12	}
13	else
14	arreglo[j+1]=arreglo[j];
15	}
16	if(j== -1)
17	arreglo[0]=aux;
18	}

Método de Inserción binaria (SHELL)

Similar al método de inserción directa, excepto que la búsqueda del orden de un elemento en la sub lista ordenada se desarrolla usando búsqueda binaria, lo que supone un ahorro de tiempo. No obstante, para la inserción de elementos sigue siendo necesario desplazar los elementos, por lo que el ahorro no se produce.

Método de Inserción Directa mejorado (SHELL)

Es un método mejorado principalmente cuando hay un gran número de elementos. En esta método no hay comparación de cada elemento con el de su izquierda, sino con el que está a un cierto

Introducción a la programación con C

número de lugares a su izquierda **llamado salto**. Dicho salto es constante y su valor inicial es $N/2$ (siendo N el número de elementos, compuesto por una división entera). Se van dando pasadas hasta que en una de estas no se intercambie ningún elemento de sitio. Entonces, el salto se reduce a la mitad y se retoman las pasadas hasta que no se intercambie ningún elemento y así sucesivamente hasta que el salto vale 1.(Facultad de Ciencias de la computación, Benemérita universidad Autónoma de Puebla. Algoritmos de ordenación)

Por ejemplo: los pasos para ordenar el siguiente arreglo son:

0	1	2	3	4	5	6	7	8 índice
9	10	4	40	21	8	25	45	55

Mediante el método de Shell serían: **n/2**= donde n es el tamaño del vector es decir n=9 Salto=9/2=4,5 , pero como debe ser entera entonces el Salto = 4

No hay cambio por que 9 es menor que 21

Primera pasada:

9	10	4	40	21	8	25	45	55
---	----	---	----	----	---	----	----	----

Se realiza el intercambio 10 por el 8 por que 10 es mayor que 8

9	8	4	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

9	8	4	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio por que 40 es menor que 45

9	8	4	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

9	8	4	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio por que 21 es menor que 55

Se divide el salto que era cuatro (4) Salto = salto/2;

Segunda Pasada:

← Realizo el intercambio 9 por el 4

9	8	4	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

4	8	9	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

4	8	9	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← Realizo el intercambio 40 por el 10

4	8	9	40	21	10	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

4	8	9	10	21	40	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

4	8	9	10	21	40	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

4	8	9	10	21	40	25	45	55
---	---	---	----	----	----	----	----	----

Se divide el salto que era dos (2) Salto = salto/2;
Salto =1

4	8	9	10	21	40	25	45	55
---	---	---	----	----	----	----	----	----

← No hay cambio

No hay cambio desde la posición donde está el número a la posición donde está el número 40 y 25 y realizo el intercambio

Introducción a la programación con C

4	8	9	10	21	40	25	45	55
---	---	---	----	----	----	----	----	----

← Realizo el intercambio 10 por el 21

El arreglo quedo completamente ordenado

4	8	9	10	21	25	40	45	55
---	---	---	----	----	----	----	----	----

Algoritmo en c:

1	int arreglo[N];
2	int salto, cambios, aux, i;
3	for(salto=N/2; salto!=0; salto/=2)
4	for(cambios = 1; cambios != 0;)
5	{
6	cambios = 0;
7	for(i=salto; i<N; i++)
8	if(arreglo[i-salto]>arreglo[i])
9	{
10	aux=arreglo[i];
11	arreglo[i]=arreglo[i-salto];
12	arreglo[i-salto]=aux;
13	cambios++;
14	}
15	}

Método Ordenación rápida (QuickSort):

Aplica el principio de "divide y vencerás" que consiste en ir subdividiendo el arreglo en arreglos más pequeños los cuales ordenará posteriormente. Para hacer esta división se toma un valor del arreglo como pivote, se mueven todos los elementos menores que este pivote a su izquierda y los mayores a su derecha. (Facultad de Ciencias de la computación, Benemérita universidad Autónoma de Puebla. Algoritmos de ordenación)

Introducción a la programación con C

A continuación se aplica el mismo método a cada una de las dos partes en las que queda dividido el arreglo.

Para dividir el arreglo se tiene que:

***Ejemplo tomado de Facultad de Ciencias de la computación, Benemérita universidad Autónoma de Puebla. Algoritmos de ordenación*

Por ejemplo, para dividir el arreglo

21	40	4	9	10	35
----	----	---	---	----	----

Los pasos serían.

21	40	4	9	10	35
----	----	---	---	----	----

Se toma como pivote el **21**. En la búsqueda de izquierda a derecha encuentra el valor 40 que es mayor que pivote y en la búsqueda de derecha a izquierda encuentra el valor 10, menor que el pivote.

Se intercambian:

21	40	4	9	10	35
----	----	---	---	----	----

{21, 10, 4, 9, 40, 35} <- Si seguimos la búsqueda, la primera encuentra el valor 40, y la segunda el valor 9, pero ya se han cruzado, así que paramos. Para terminar la división, se coloca el **pivote** en su lugar (en el número encontrado por la segunda búsqueda, el **9**, quedando:

{9, 10, 4, 21, 40, 35} <- Ahora tenemos dividido el arreglo en dos arreglos más pequeños: el {9, 10, 4} y el {40, 35}, y se repetiría el mismo proceso.

Algoritmo en c:

```
1 #include <stdio.h>
2 void ordenar(int *arreglo, int desde, int hasta)
3 {
4     int i, d, aux;
5     if(desde >= hasta)
6         return;
7     for(i=desde+1, d=hasta;;)
8     {
9         for(; i<=hasta&&arreglo[i] <= arreglo[desde]; i++);
10        for(; d>=0 && arreglo[d] >= arreglo[desde]; d--);
11        if(i<d)
12        {
13            aux=arreglo[i];
14            arreglo[i]=arreglo[d];
15            arreglo[d]=aux;
16        }
17    Else
18        break;
19    }
20    if(d ==desde-1)
21        d = desde;
22    aux=arreglo[d];
23    arreglo[d]=arreglo[desde];
24    arreglo[desde]=aux;
25    ordenar(arreglo, desde, d-1);
26    ordenar(arreglo, d+1, hasta);
27 }
28
29 void main() {
```

Introducción a la programación con C

30	ordenar(arreglo, 0, N-1);
31	}

En C hay disintas librerías que implementan varios algoritmos, por ejemplo *stdlib.h* que contiene la función *qsort()* encargada de implementar el algoritmo de quicksort descrito:

1	#include <stdio.h>
2	#include <stdlib.h>
3	
4	int funcion(const void *a, const void *b)
5	{
6	if(*(int *)a<*(int *)b)
7	return(-1);
8	else if(*(int *)a>*(int *)b)
9	return(1);
10	Else
11	return(0);
12	}
13	
14	void main()
15	{
16	qsort(arreglo, N, sizeof(arreglo[0]), funcion);
17	}

qsort(nombre_arreglo, numero, tamaño, función);

El campo nombre_arreglo indica el arreglo a ordenar, numero proporciona el total de elementos del arreglo, tamaño indica el tamaño en bytes de cada elemento y función es una función a implementar, esta recibe dos elementos y devuelve 0 si son iguales, algo menor que 0 si el primero es menor que el segundo y algo mayor que 0 si el segundo es menor que el primero.

Claramente, es mucho más cómodo usar qsort que implementar toda la función, pero hay que tener mucho cuidado con el manejo de

Introducción a la programación con C

los punteros en la función, sobre todo si se está trabajando con estructuras.

Cadenas

En los arreglos se definen como un conjunto de caracteres con un carácter de terminación nulo ("\\0"). El hecho de que la cadena deba terminar en un carácter nulo significa que para declarar un vector de caracteres es necesario que el tamaño sea más largo.

Declaración

Tipo de las variables nombre[cantidad de elementos];

Por ejemplo:

char str[50] ;

Gráficamente se puede observar así:

Índice	0	1	2	3	4	5	6	7
	G	U	S	T	A	V	0	\\0

La definición anterior se puede declarar así:

```
char nombre[] = "GUSTAVO";
```

Se está definiendo una cadena de 8 elementos y el compilador, automáticamente inserta el carácter Nulo "\\0", indicando el final del arreglo.

Para trabajar con cadenas utilizamos la **función gets ()**, esta función utiliza la librería STDIO.H. Lee caracteres hasta que pulse la tecla enter, el retorno de carro no se almacena, pero se reemplaza por un carácter nulo.

Ejemplo:

Printf ("introduzca el nombre no mayor de 50 caracteres"):

```
gets(str);  
printf ("%s",str);
```

La biblioteca estándar de C suministra muchas funciones relacionadas con cadenas, pero las funciones más importantes son:

strcpy (hacia, desde)	"Estas funciones requieren un archivo de cabecera
strcat (hacia, desde)	#include <string.h>
strlen (nom_cadena)	
strcmp (cadena1, cadena2)	

La función strcpy (cad1, cad2)

Esta función copia el contenido desde cadena2 hacia la cadena1, el contenido de cadena2 no cambia, esta función no realiza comprobación de límite, por lo tanto debemos asegurarnos que la cadena1 sea lo suficientemente grande para contenerlo.

Ejemplo:

```
char cadena [20];
strcpy(cadena, "hola") //Copia "hola" en una cadena
printf ("%s", cadena) //Imprime la cadena
```

La función strcat (cad1, cad2)

La función **strcat (cad1, cad2)** concatena una copia de *cad2* en *cad1* añadiéndola al final de *cad1* y dejando un carácter nulo.

El carácter nulo de terminación, que originalmente tenía *cad1* es sustituido por el primer carácter de *cad2*. La cadena *cad2* no se toca en esta operación. La función **strcat ()** devuelve *cad1*.

```
char cadena [20];
// Copia la palabra hola en una cadena:
strcpy(cadena, "hola")
//Concatena la palabra hola con la palabra que tal:
strcat(cadena, "que tal")
//Imprime la cadena la siguiente palabra hola que tal.
printf ("%s", cadena)
```

La función strcmp (cad1, cad2)

La función **strcmp (cad1, cad2)** compara lexicográficamente dos cadenas que finalizan con el carácter nulo y devuelve un entero que se interpreta de la siguiente forma:

X = strcmp(cadena1,cadena2);

Introducción a la programación con C

- Si (**x < o**) Las cadenas no son iguales la cadena 1 es menor que la cadena2.
- Si (**x > o**) Las cadenas no son iguales la cadena 1 es mayor que la cadena2.
- Si (**x == o**) Las cadenas son iguales

Valor	Interpretación
X = Menor que 0	<i>cad1</i> es menor que <i>cad2</i>
X = 0	<i>cad1</i> es igual a <i>cad2</i>
X = Mayor que 0	<i>cad1</i> es mayor que <i>cad2</i>

Ejemplo

```
printf ("%d", strcmp (uno, uno)); //Comparación de las cadenas  
l = strcmp (uno, uno);
```

La variable l puede tomar el valor de cero, valor mayor que cero y un valor menor cero.

La función strlen ()

Devuelve la longitud de una cadena, en carácter su forma general es:

```
char cadena [20];  
int longitud;  
strcpy(cadena, "guillermo"); //Copia el nombre en una cadena.  
longitud = strlen(cadena); //Devuelve la longitud de la cadena.  
printf("%d", longitud ); //Imprime la longitud de la cadena.
```

Matrices:

El lenguaje C soporta, a diferencia de otros lenguajes, arreglos de n dimensiones.

Los arreglos de 3 o más dimensiones no son muy comúnmente usados debido a la cantidad de memoria que se necesita para su almacenamiento. Otro de los motivos por los cuales su uso es muy restringido, obedece al hecho de que se necesita mucho tiempo para el acceso a un elemento específico, lo que hace mucho más lento el acceso a un arreglo multidimensional que a un arreglo unidimensional con el mismo número de elementos. Los arreglos multidimensionales más usados son los de dos dimensiones, más conocidos como matrices.

Declaración de una matriz

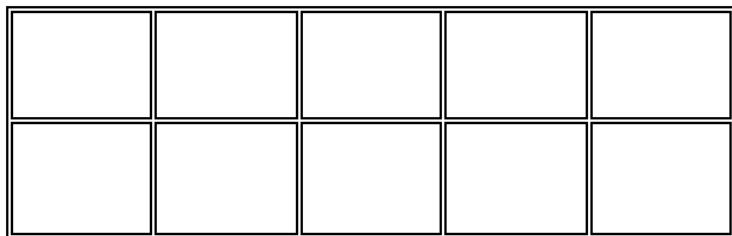
Tipo dato nombre_matriz [cant_elementos] [cant_elementos];

Por ejemplo:

int M[2][5];	→	Declaración de matrices enteras
float numeros[20][6];	→	Declaración de matrices decimales
long double cantidades[2][5];	→	Declaración de matrices para cantidades grandes

Representación gráfica de una matriz:

Columnas



Filas

Una matriz de 2×5 seria: entero x [2] [5] // filas y columnas
Los elementos se numeran y referencian con un índice.

Definición de matrices

Las matrices en lenguaje C pueden ser definidas como los arreglos, pero existen algunas diferencias en cuanto a la sintaxis que se debe utilizar, por ejemplo: para definir una matriz de 3 filas y 4 columnas se puede escribir la siguiente instrucción:

int m [3] [4];

Donde cada elemento puede contener un número de tipo entero.
Gráficamente esta matriz **M** se puede definir así:

$m[3][4]$

		Columnas			
		0	1	2	3
Filas	0	0 , 0	0 , 1	0 , 2	0 , 3
	1	1 , 0	1 , 1	1 , 2	1 , 3
	2	2 , 0	2 , 1	2 , 2	2 , 3

Índice

Inicialización de una matriz

Veamos la siguiente sentencia:

```
for (i=0; i<3; i++)
{
    for (j=0; j<4; j++)
    {
        m [i][j] = 10;
    }
}
```

En esta sentencia se está variando un índice **i** desde 0 hasta 3 y un índice **j** desde 0 hasta 4. En cada variación se está asignando al elemento 10 en las posiciones de la matriz **i, j**. El efecto es asignar a cada elemento de una matriz 3 por 4 el número 10.

Ejercicios de matrices

1.Llenar una matriz 3x5 y determinar cuántos números negativos y positivos contiene la matriz.

2.Llenar una matriz 4x4 y determinar el número mayor y el menor que se encuentra en la matriz

3.Buscar un elemento en la matriz.

4.Imprimir los números del 10 al 1.

5.Imprimir la suma de los números del 1 al 10.

6.Imprimir los cuadrados de los números del 1 al 10

7.Imprimir la suma de los cuadrados de los números del 1 al 10

8.Construir una función que reciba como parámetro un número N, y calcule la suma de todos los enteros menores que el número recibido.

9.Construir un programa que dados dos enteros M y N diferentes, calcule la suma de los cuadrados de los números que hay entre ellos, sin incluirlos.

Introducción a la programación con C

10. Hacer una función que reciba 2 números y que imprima en una tabla los números que hay entre el primer número y el segundo. Frente a cada número debe aparecer su cuadrado, su cubo, su raíz cuadrada y su raíz cúbica. Ejemplo:

Ítem	Número	Cuadrado	Cubo	Raíz cuadrada	Raíz cúbica
1	10	100	1000	3.166....	2.154....
.....				
91	100	10000	1000000	10	4.6415...

11. Escriba un programa que calcule la suma de los números que existen entre dos números dados. Debe considerar ambos números.

12. Escriba una función que calcule cuántos números naturales hay entre 2 números dados.

13. Escriba un programa que calcule el valor promedio de los números naturales que existen entre dos números dados. Debe considerar ambos números. (Use las dos funciones anteriores)

14. Hacer un programa que lea 10 números e imprima el cuadrado y el cubo de cada número. Los números se deben leer dentro de la función.

15. Hacer un programa que reciba un número.

Si el número es menor o igual que 5: imprime los números del 1 al 5.

Si el número es mayor que 5 y menor o igual que 10: imprime los números del 5 al 10.

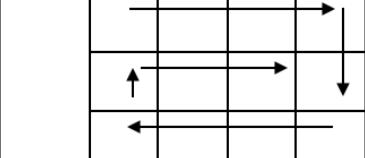
16. Si el número es mayor que 10: imprime los números del 10 al 20.

Problema No 1.

Realizar el siguiente programa. Se lee una matriz de tipo entero n*m

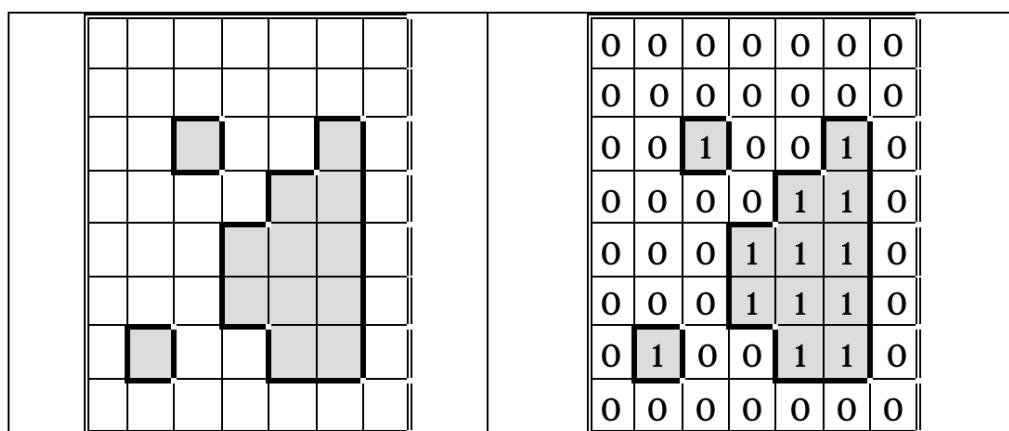
9	5	3	6
9			
10	4	5	2
12	1	33	3

El programa debe realizar el ordenamiento de menor a mayor de tal forma que los datos se ubiquen teniendo en cuenta el concepto de **espiral**.

				1	2	3	3	
				12	33	99	4	
				10	6	5	5	

Problema No 2.

Un terreno se encuentra dividido en parcelas de las cuales unas de ellas son cultivadas y otras no. Dicho terreno es plano y su forma corresponde a un cuadrado. Las parcelas cultivadas como no cultivadas corresponden de igual manera a áreas pequeñas que son cuadradas y todas de igual tamaño. Las parcelas cultivadas pueden o no ser colindantes, como se indica en la figura.



Dado que: **El área de cada parcela es de 10m².**

El terreno corresponde a una matriz de $n \times m$ dimensiones. Las parcelas cultivadas se representan por una entrada de la matriz con un valor igual a 1 y las no cultivadas con el valor 0.

Calcular **el área total de terreno cultivado, el perímetro del área cultivada y el área de terreno no cultivado.**

a)

Área Cultivada

$$(10\text{m}^2 \cdot 13) = 130\text{m}^2.$$

Área no cultivada

$$(10\text{m}^2 \cdot 43) = 430\text{m}^2.$$

b)

Perímetro

$$(3.17 \text{ m} \cdot 24) = 76.08 \text{ m}$$

Problema No 3.

Llenar una matriz de 6x6 determinar.

- a. cuantas veces se repite el mismo número dentro de las matrices
- b. borrar un elemento de la matriz.
- c. Mostrar Los múltiplos de 3.
- d. Ordenar la matriz por cual quiera de los métodos estudiados.
- e. Realizar la sumatoria de primos de la matriz por las diagonales
- f. Crear otra matriz y realizar la multiplicación de matrices
- g. Hallar el menor primo y mayor primo de la matriz

Problema No 4.

Escriba un programa que realice la versión informática del Ahorcado (juego del ahorcado) Se trata de adivinar que palabra se trata, introduciendo las letras cada vez que se introduzca la letra, se examina se examina la palabra mágica para verificar si es correcta o no. Guarde la cuenta del número de letras introducidas para completar la palabra, el jugador gana cuando completa la palabra.

Problema No 5.

Si se necesita guardar la información relacionada con el tablero de un juego de tic tac toe (el tradicional triqui), se puede declarar la siguiente matriz:

tablero : matriz [3][3] de carácter

X		
O	X	O
O	O	X

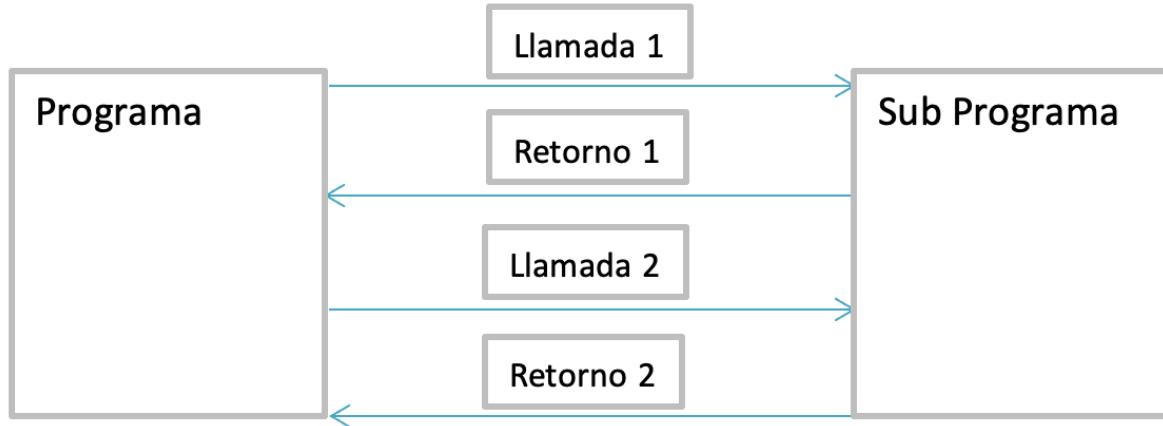
Capítulo 4

Capítulo 4: Funciones y estructuras en C

En este capítulo se presentará el concepto de función o subprograma, una herramienta que ayudará a mantener un trabajo limpio, ordenado y sencillo de entender.

Después de esto se explicará el concepto de estructuras, un tipo de dato abstracto creado por el usuario con el fin de hacer la programación más útil y rápida a la hora de desarrollar un proyecto.

Funciones



Una función es un conjunto de líneas de código, que realizan una tarea específica y puede retornar un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir la cantidad de código, cuando una función es invocada se le pasa el control a la misma, una vez que esta finalizó con su tarea el control se devuelve al punto desde el cual la función fue llamada.

Estructuras o registros

Por definición la matriz obliga que todos sus datos sean del mismo tipo, es decir, no se permite la mezcla entre carácter y un entero por ejemplo.

Una **estructura**, según el escrito de 1^a parte: *Metodología de Programación*, es una forma de agrupar un conjunto de datos de distinto tipo bajo un mismo nombre o identificador.

Su forma general es mediante la palabra **registro o estructura**:

```
struct nombre_estructura
{
    tipo datos;
    tipo datos;
    ....
};
```

La expresión **tipo datos**; generaliza la forma de indicar los campos que tendrá la estructura. En ese mismo orden, para declarar **variables de tipo estructura o registro** se emplea la siguiente notación:

```
struct nombre_structura <nombre_variable>
```

Por ejemplo, si se desea diseñar una **estructura** que guarde los **datos** correspondientes de una **video tienda**, esta estructura a la que se le llamará **video 200**, deberá guardar: los datos del usuario, datos de la película y un calendario.

Datos del usuario

Nombre, dirección, teléfono, cedula.

Datos de la película

El nombre de la película, el código de la película, género de la película

Datos del calendario

Día, mes, año, hora

Fecha de alquiler. Fecha de recibido.

Cada uno de estos datos se denomina **campo o miembro** de la estructura. El modelo de esta estructura puede crearse del siguiente modo:

```
struct usuario{  
    char nombre[31]; // cadena de caracteres  
    char direccion[21];  
    long int cedula; // número entero largo  
    int telefono;  
};
```

El código anterior crea el **tipo de dato usuario** “nombre de la estructura. Obsérvese la necesidad de incluir un carácter (;) después de cerrar las llaves, hay que recordar que es un separador de sentencias.

Para declarar una **variable de tipo usuario** se debe utilizar la palabra **estructura (struct)** y el nombre que se le ha asignado dado (usuario):

Declaración de la variable de la estructura.

```
struct usuario registro;
```

Para acceder y modificar la información que se almacenará en los miembros de una estructura se utiliza el operador **punto (.)**, precedido por el **nombre de la variable** y seguido del nombre del campo **variable.campo**.

Por ejemplo, para dar valor al teléfono del usuario, el valor 903456 se escribirá:

```
registro.telefono = 903456;
```

y para guardar la dirección de este mismo usuario, se escribirá:

```
registro.direccion = "C/ Rios Rosas 1,2-A";
```

De esta forma se puede ver el contenido de un campo del registro, si se escribe:

```
x = registro.telefono
```

Al hablar de registros tenemos en cuenta una cierta cantidad de elementos, por lo que no es muy conveniente el uso de 1 solo registro almacenado en una sola variable. De lo anterior, una forma inmediata de mantener múltiples registros es mediante el empleo de una **una matriz de registros**.

Ejemplo estructura.

Declaración de los nombres de las estructuras

1	#include <stdio.h>
2	#include <conio.h> //curses para usuarios linux
3	#include <string.h>
4	#include <stdlib.h>
5	
6	struct usuario {

Introducción a la programación con C

```
7 char nombre[30];
8 char sexo [9];
9 int cedula[12];
10 char direccion[30];
11 };
12 struct pelicula {
13     char nombre_pelicula[30];
14     char genero [9];
15     int codigo[12];
16     char direccion[30];i
17 };
18
19 struct calendario {
20     char mes[30];
21     char anho [9];
22     int dia[12];
23     char fecha_alquiler[30];
24     char fecha_entrega[30];
25 };
26
27 void main() {
28     clrscr();
29     int i,j,k,p;
30     float dato;
31     char temp;
32
33     struct usuario registro;
34     struct pelicula video;
35     struct calendario tiempo;
36     printf("digite el nombre");
37     gets (registro.nombre);
```

Introducción a la programación con C

38	printf("digite el cedula");
39	gets(registro.cedula);
40	printf("digite la fecha");
41	
42	scanf("%f",&dato);
43	tiempo.dia=dato;
44	printf("digite el nombre de la pelicula");
45	gets(video.nombre_pelicula);
46	
47	printf ("\n Nombre: %s",registro.nombre);
48	printf ("\n cedula: %s ",registro.cedula);
49	printf ("\n Nombre _ pelicula: %s",video.nombre_pelicula);
50	getch();
51	}

Estructuras anidadas

Se producen cuando algún miembro de la estructura es a su vez otra estructura.

Ejemplo

1	struct fecha {
2	int dia;
3	int mes;
4	int año;
5	};
6	struct persona {
7	char nombre[20];
8	struct fecha nacimiento;
9	};
10	struct persona p;

Arrays de estructuras

Se define primero la estructura y luego se declara una variable de tipo estructura como un arreglo.

Ejemplo:

```
struct registro {  
    char apellido[10];  
    char nombre [10];  
    int cedula[10];  
};  
  
struct registro fami[100];
```

Para acceder a una determinada estructura se indexa el nombre de la estructura:

```
fami[2].apellido = "pepito";  
fami[1].nombre = "perez";  
fami[4].cedula = 76318649;
```

Paso por valor de miembros de una estructura a una función.

El paso por valor se realiza tal como lo realizamos con una variable simple

```
void f1(int); //declaración el prototipo de función  
f1(a.codigo); //Llamada a la función con un miembro de la función  
void f1(int x); //definición de la función
```

Paso por dirección de un miembro de la estructura a una función.

Se realiza como si fueran variables simples. Por ejemplo, para pasar por valor el miembro a.codigo

```
void funcion f1(int *) //declaración el prototipo de función  
f1(&a.codigo); //llamada a la función  
void f1 (int *x) // definición de la función  
{..... acciones .....
```

Hay que tener en cuenta, que si lo que se pasa a una función es un miembro de una estructura que sea un arreglo, éste siempre pasa el nombre del vector por dirección (ya que el nombre de un arreglo es la dirección del primer elemento del mismo).

Paso por valor de estructuras completas a funciones.

En el siguiente ejemplo, suma es una función que recibe dos estructuras pasadas por valor y a su vez devuelve un valor a una estructura.

```
struct vector {  
    int x, y, z;};  
  
int struct vector (struct vector m1 , struct vector m2 );  
  
void main() {  
    struct vector v1,v2,v3;  
    ...  
    v3=suma(v1,v2); ...}  
  
int struct vector suma(struct vector m1, struct vector m2){  
    m1.x+=m2.x;  
    m1.y+=m2.y; m1.z+=m2.z; return (m1.z);  
}
```

Paso por dirección de estructuras completas a funciones.

Cuando las estructuras son de gran tamaño, es mejor en términos de eficiencia pasarlas por dirección (direcciones de memoria). Se utilizan punteros a estructuras para realizar la comunicación. Para acceder a los miembros de la estructura debe utilizarse la combinación de los operadores * y punto, sin embargo el operador punto tiene más precedencia que *, siendo necesario el uso de paréntesis para asegurar que se aplique primero * y después punto. También se puede usar el **operador** -> para acceder a los miembros de una estructura referenciada por un puntero.

1	#include <stdio.h>
2	struct pareja {
3	int a, b;
4	};
5	
6	void f1(struct pareja *q) {

Introducción a la programación con C

7	q->a++; /* equivalente a (*q.a)++ pero más usado */
8	q->b++;
9	}
10	
11	void main() {
12	struct pareja p = {3,17} //inicialización
13	f1(&p); // Llamada a la estructura por referencia
14	printf ("valor de a:%d valor de b:%d\n",p.a,p.b);
15	/* escribe 14.y 18 */
16	}

PUNTEROS A ESTRUCTURAS

Declaración: struct dir * pruntero_dir;

Existen dos usos principales de los punteros a estructuras:

- Para pasar la dirección de una estructura a una función.
- Para crear listas enlazadas y otras estructuras de datos dinámicas.

Para encontrar la dirección de una variable de estructura se coloca & antes del nombre de la estructura.

Ejemplo:

```
struct bal {  
    float balance;  
    char nombre[80];  
} persona;  
struct bal *p;
```

```
p = &persona; //coloca la dirección de la estructura persona en el  
puntero p
```

No se puede usar el operador punto para acceder a un elemento de la estructura, a través del puntero a la estructura. Debemos utilizar el operador **flecha ->**

p -> balance.

Definición de nuevos tipos de datos

Introducción a la programación con C

La creación de nuevos nombres de tipos de datos se realiza utilizando la palabra reservada **typedef, union**:

La palabra **typedef** crea un sinónimo del tipo de dato utilizando el siguiente formato:

```
typedef tipodato nombre
```

Ejemplos:

Con tipos simples:

```
typedef int ENTERO  
typedef float REAL
```

```
ENTERO a, b; // Declara variables a y b de tipo entero  
REAL c;
```

Con tipos estructurados:

```
typedef struct{  
    int dia;  
    int mes;  
    int anio;  
} FECHA;  
  
FECHA a;
```

Se trabaja igual que la estructura, la única diferencia es que se omite la palabra **struct** y ahora solo se trabaja con **FECHA** para declarar una variable de la estructura.

Enum

Es una estructura que permite la enumeración de un conjunto de elementos, dicha numeración es escogida exclusivamente por el programador. Las constantes representan los valores que pueden ser asignados a las variables declaradas del tipo del enum.

enum nombre {val1,val2,...,valn}; donde val_i es un identificador de constantes.

Si no se les asigna ningún valor, se inicializan con los valores enteros 0, 1, 2, ...

Introducción a la programación con C

Se pueden asignar otros valores indicándolos en la enumeración

```
enum nombre {val1=0,val2=10,val3=13,...};
```

Puede suceder que más de una constante de enumeración tenga el mismo valor entero.

```
enum color{rojo=-1,azul,amarillo,verde,negro =0};
```

- Las variables de enumeración pueden utilizarse como enteros; asignándoles valores, compararlas, pero son utilizadas sólo internamente.
- No se puede leer una variable de tipo enum, se puede leer un entero y asignárselo a una variable.
- No se puede escribir más que el valor entero de la variable enum.
No se puede escribir su nombre.
- Los tipos enumerados no aportan capacidades nuevas al lenguaje, pero aumentan la claridad de los programas.

Ejercicios Propuestos

1.Realizar un programa para una guía telefónica, que permita guardar el número telefónico, el nombre y la dirección de una persona.

2.Realizar un programa para un hospital que permita saber los datos de un paciente. Nombre, edad, sexo y seguro social.

3.Realizar un programa que permita saber al cliente el valor del producto, cuando digite su código.

4.Realizar una función para una agencia de viajes, el cual permita organizar los cupos según el estrato, ubicando 20 personas por estrato.

Problema 1.

Escribir un programa que ingrese datos hasta que se introduce un número negativo y calcule:

- 1.Llenar //procedimiento
- 2.Mostar //procedimiento
- 3.La media. // función
- 4.EL máximo. // función

Introducción a la programación con C

5.EL mínimo. //función
6.Contar el número de primos //en el procedimiento incluir la función booleana que diga si es primo o no
7.Colar los pares en una lista_1 y los impares en lista_2 //procedimiento

Problema 2.

Realizar la siguientes funciones o procedimientos con un menú (estructura case).....

```
Potencia (base exp);  
Valor_Absoluto (a);  
Cuadrado (x);  
Potencia (Valor_Absoluto (a) , exp);  
Suma (Potencia(base, exp), Cuadrado(x) ) ;  
Menu(.....);  
  
//ciclo do {  
    Switc (op)  
  
}while( )  
  
int main(){  
    //la menor cantidad de líneas de código  
}
```

Problema No 3.

Se desea escribir un programa para establecer el valor que debe pagar por Consulta Médica cada uno de los pacientes afiliados al régimen subsidiado en un Centro Salud. El gobierno ha establecido que a partir de este mes cada uno de los afiliados pague un bono de acuerdo con el salario devengado. Además se estableció que a la tarifa de los estratos 1 y 2 se le aplica un descuento el cual será cubierto por Fisalud.

Los datos de cada paciente son los siguientes:

Cédula.

Nombre.

Estrato.

Salario devengado.

Tarifa: Se ha establecido la tarifa de acuerdo al estrato al que pertenecen los afiliados de la siguiente forma, Estrato 0= \$0, Estrato 1= \$ 800, Estrato 2 = \$ 1200, Estrato 3 = \$2000.

Introducción a la programación con C

Calcular el Total de la consulta y retornar su valor utilizando una función de acuerdo a las siguientes condiciones.

Total consulta = Tarifa + bono – descuento

El valor del bono se calcula de la siguiente manera:

Un 0.5 % del salario devengado, si el afiliado es de estrato 0.

Un 0.8 % del salario devengado, si el afiliado es de estrato 1.

Un 1% del salario devengado, si el afiliado es de estrato 2.

Un 1.2 % del salario devengado, si el afiliado es de estrato 3.

El descuento se aplica a los estratos I y II de la siguiente manera

El 50% de la tarifa, si el salario devengado es menor o igual a 500.000 y el paciente es de estrato 1.

El 40% de la tarifa, si el salario devengado es mayor a 500.000 y el paciente es de estrato 1.

El 30% de la tarifa, si el salario devengado es menor o igual a 500.000 y el paciente es de estrato 2.

El 25% de la tarifa, si el salario devengado es mayor a 500.000 y el paciente es de estrato 2. Imprimir los datos de los pacientes junto con total de la consulta calculada.

Capítulo 5

Capítulo 5: Manejo de archivos

En este capítulo se explicarán los conceptos básicos relacionados con el manejo de archivos: creación, lectura, escritura, apertura y muchas funciones que ayudarán a desarrollar aplicaciones con un mayor nivel de complejidad y utilidad.

Manejo de archivos

Archivos

Los archivos son un conjunto simple de almacenamiento que permiten a la computadora distinguir entre los diversos tipos de información. Aunque no siempre es el caso, un archivo se suele encontrar en un formato legible por los usuarios, aun así en un archivo se agrupan instrucciones, números, palabras o imágenes en unidades coherentes que el usuario puede recuperar, modificar, eliminar, guardar o enviar a un dispositivo de salida.

Se puede decir que un archivo es un conjunto completo de información identificado con un nombre, puede ser un programa, un conjunto de datos utilizados por el programa o un documento creado por los usuarios.

Es importante indicar que los archivos no son únicamente los archivos que se guardan en el disco duro, en C y muchos otros lenguajes los dispositivos del computador se tratan de como archivos: la impresora, el teclado, la pantalla,...etc.

Manejo de archivos en lenguaje C

El lenguaje C proporciona un acceso secuencial y directo a los registros de un archivo, pero no soporta el acceso indexado a un registro dado.

Los archivos en C se pueden clasificar según la información que contengan, en dos grupos:

- Archivos de Texto
- Archivos Binarios.

Los archivos de texto se componen de una serie de caracteres organizados en líneas que terminan por un carácter '\n' y permiten el uso de la línea siguiente, esto hace pensar en la idea de usar la impresora como si fuese un archivo de texto.

Por otro lado los **archivos binarios** constan de una secuencia de bytes, por tal razón podemos decir que cualquier archivo que no sea de texto, será binario.

A la hora de trabajar con archivos, tendremos que especificar antes de usarlos, si serán de texto o binarios.

El lenguaje C trata a los archivos como punteros, en realidad un archivo es un puntero a una estructura de nombre predefinido **FILE**,

Introducción a la programación con C

cuyas componentes son las características de la variable archivo declarada, cada archivo deberá tener una estructura FILE asociada.

La estructura **FILE** se encuentra definida en el archivo de cabecera **stdio.h**, con lo cual es necesario incluirla en todos los programas que trabajen con archivos.

```
#include <stdio.h>
```

La forma de declarar variables de tipo FILE es la siguiente:

```
FILE *X, *Y,...
```

Se estudiarán los distintos modos en que podemos abrir un archivo, así como las funciones para leer y escribir en él.

Apertura: fopen();

Esta función cierra el archivo apuntado por el puntero y reasigna este puntero a un archivo que será abierto. Su sintaxis es:

```
fopen(nombre del archivo,"modo de apertura");
```

Donde **nombre del archivo** es el nombre del nuevo archivo que se quiere abrir, luego el **modo de apertura** y finalmente el puntero que va a ser reasignado.

Antes de abrir un archivo es necesario declarar un puntero de tipo **FILE**. Para abrir el archivo utilizaremos la función **fopen()**.

Su sintaxis es:

```
FILE *puntero;
```

```
puntero = fopen (nombre del archivo, "modo de apertura" );
```

Donde **puntero** es la variable de tipo **FILE**, **nombre del archivo** es el nombre que se le da al archivo que se va a crear o abrir, este nombre debe ir encerrado entre comillas. También se especificará la ruta donde se encuentra el nombre del archivo (En este caso no se pondrán las comillas).

Ejemplo

```
puntero = fopen("DATOS.DAT","r");
puntero = fopen("C:\\TXT\\SALUDO.TXT","w");
```

Introducción a la programación con C

Un archivo puede ser abierto en dos modos diferentes, en modo texto o en modo binario.

A continuación se observa con más detalle.

Modo texto

wt	Crea un archivo de escritura. Si ya existe lo crea de nuevo.
w+	Crea un archivo de lectura y escritura. Si ya existe lo crea de nuevo.
a	Abre o crea un archivo para añadir datos al final del mismo.
a+	Abre o crea un archivo para leer y añadir datos al final del
r	Abre un archivo de lectura.
r+	Abre un archivo de lectura y escritura.

Modo binario

Wb	Crea un archivo de escritura. Si ya existe lo crea de nuevo.
W+b	Crea un archivo de lectura y escritura. Si ya existe lo crea de nuevo.
Ab	Abre o crea un archivo para añadir datos al final del mismo.
A+b	Abre o crea un archivo para leer y añadir datos al final del
Rb	Abre un archivo de lectura.
R+b	Abre un archivo de lectura y escritura.

Comprobar si está abierto

Es muy importante comprobar si un archivo realmente está abierto luego de intentar abrirlo. Pueden ocurrir errores y fallos en el sistema: el archivo no existe o no se encuentra la ruta especificada, puede estar dañado o no tener permisos de lectura.

Si se intenta hacer operaciones sobre un puntero tipo FILE cuando no se ha conseguido abrir, el archivo puede tener problemas.

Si el archivo no se ha podido abrir, el puntero archivo (puntero a FILE) tendrá un valor NULL. Si se ha abierto con éxito tendrá un valor distinto. Por ende para corroborar si existen errores nos debemos fijamos en el valor del puntero:

```
FILE *ARCHIVO;  
if (archivo==NULL)  
{
```

Introducción a la programación con C

```
printf( "No se puede abrir el archivo.\n" );
exit(1);
}
```

Si la condición `archivo==NULL` se evalúa como verdadero, significa entonces que no se ha podido abrir por algún error, en este caso lo más conveniente es salir del programa. Para salir se utiliza la función `exit(1)`, el 1 indica al sistema operativo que se han producido errores.

Cierre

Una vez finalizado el trabajo con un archivo es recomendable cerrarlo. Los archivos se cierran al finalizar el programa, por otra parte el número de archivos que pueden estar abiertos es limitado. Para cerrar los archivos se usa la función

`fclose();`

Esta función cierra el archivo cuyo puntero le indicamos como parámetro, si el archivo se cierra con éxito devuelve 0.

```
fclose(puntero);
```

Un ejemplo:

```
FILE pf; /*Crea un apuntador de tipo archivo
//Crea un archivo AGENDA.dat de modo lectura y binario
pf=fopen("AGENDA.dat","rb");
//Si el apuntador es NULL entonces el archivo está vacío
if (pf != NULL)
{
    acciones
}
else {
    fclose(pf); //Cierra el archivo.
}
```

Errores típicos manejando archivos

Las funciones del grupo de `fopen` son:

`fprintf, fscanf, fwrite, fread, fclose`

Constituyen una interfaz de más alto nivel para el manejo de archivos que las de la familia de `open` (`read, write, close, etc`), por lo

Introducción a la programación con C

que deben utilizarse las primeras, salvo que haya una buena causa para utilizar las segundas.

No comprobar si el archivo se ha abierto con éxito

```
if ((file = fopen("c://archivo.txt","r")) == NULL);  
fprintf ("Error de apertura de archivo\n");
```

No vaciar los búferes.

Cuando se escribe un dato en un archivo mediante **fwrite** o **fprintf** los datos no se guardan en el archivo hasta que se llena un buffer interno o se cierra el archivo (cosa que ocurre automáticamente al terminar el programa, sí todo ha ido bien).

Al utilizar los datos guardados en un archivo antes de terminar el programa, se debe llamar a **fflush** para forzar el vaciamiento de los bufferes.

No cerrar los archivos.

Otra razón para cerrar un archivo que no se esté utilizando es que el máximo número de archivos que un proceso puede tener abiertos está limitado y cuando se intente abrir un archivo más de los permitidos la operación fallará.

Funciones para trabajar con archivos

FUNCION				
fopen	fclose	fgets	fputs	fgetc
fseek	ferror	eof, feof	ftell	fputc

Lectura del fichero - getc

Esta herramienta permite leer caracteres uno a uno dentro de un fichero. Se puede usar también la función fgetc (son equivalentes, la diferencia es que getc está implementada como macro), además de ellas existen otras herramientas (funciones) como fgets, fread que leen más de un carácter y que se estudiarán más adelante.

El formato de la función getc (y de fgetc) es:

Introducción a la programación con C

```
int getc(FILE *fichero);
```

Comprobar fin de fichero - feof

Al entrar en el bucle while, la lectura se realiza hasta que se encuentre el final del fichero. Para detectar el final del fichero se pueden usar dos formas:

- Con la función **feof()**
- Comprobando si el valor de *letra* es **EOF**.

Esta función comprueba si se ha llegado al final del *fichero*, en cuyo caso devuelve un valor distinto de 0, si no se ha llegado al final de fichero devuelve un cero. Por eso se usa del siguiente modo:

```
while ( feof(fichero)==0 )
ó
while ( !feof(fichero) )
```

La segunda forma consiste en comprobar si el carácter leído es el de fin de fichero

EOF:

En otras palabras hasta que sea fin de archivo
while (letra!=EOF)

Cuando se trabaja con ficheros de texto no hay ningún problema, pero si se está manejando un fichero binario se encontrará EOF antes del fin del fichero, por eso es mejor usar feof.

Cerrar el fichero - fclose

Luego de haber realizado todas las operaciones deseadas sobre el fichero, se debe de cerrar este ultimo. Es importante no olvidar este paso, pues el fichero podría corromperse. Al cerrarlo se vacían los buffers y el fichero se guarda en el disco. Mediante la función **fclose(fichero)** se logra cerrar el fichero, si todo va bien fclose retornará un cero, si existen problemas retorna otro valor. Estos problemas se pueden producir si el disco está lleno, por ejemplo:

```
if (fclose(fichero)!=0)
printf( "Problemas al cerrar el fichero\n" );
```

Lectura de líneas – fgets

La función **fgets** es muy útil para leer líneas completas desde un fichero. El formato de esta función es:

```
char  
fgets(char *buffer, int longitud_max, FILE *fichero);
```

Esta función lee desde el fichero hasta que encuentra un carácter '\n' o hasta que lee longitud_max-1 caracteres y añade '\0' al final de la cadena. La cadena leída la almacena en *buffer*.

Si se encuentra EOF antes de leer ningún carácter o si se produce un error la función devuelve NULL, en caso contrario devuelve la dirección de *buffer*.

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	
5	int main() {
6	FILE *fichero;
7	char texto[100];
8	Fichero = fopen("origen.txt","r");
9	if (fichero == NULL) {
10	printf("No se puede abrir el fichero.\n");
11	exit(1);
12	}
13	printf("Contenido del fichero: \n");
14	fgets(texto,100,fichero);
15	while (feof(fichero) == 0) {
16	printf("%s", texto);
17	fgets(texto,100,fichero);
18	}
19	if (fclose(fichero) != 0)
20	printf("Problemas al cerrar el fichero.\n");

21 }

Ahora se analizará un ejemplo que lee un fichero de texto y lo muestra en la pantalla y donde se utilizan todos los elementos que hasta el momento se han tratado:

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	
5	int main() {
6	FILE *fichero;
7	char letra;
8	fichero = fopen("origen.txt","r");
9	if (fichero == NULL) {
10	printf("No se puede abrir el fichero.\n");
11	exit(1);
12	}
13	printf("Contenido del fichero: \n");
14	letra = getc(fichero);
15	while (feof(fichero) == 0) {
16	printf("%c", letra);
17	letra=getc(fichero);
18	}
19	if (fclose(fichero)!= 0)
20	printf("Problemas al cerrar el fichero\n");
21	}

Escritura de ficheros

La escritura en ficheros se explicará con un ejemplo en donde se abrirá un fichero 'origen.txt' y se copiará en otro fichero 'destino.txt', además el fichero se muestra en pantalla. Las partes nuevas están marcadas en negrita para que se vea la diferencia entre este y el último ejemplo ya visto:

Introducción a la programación con C

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	
5	int main() {
6	FILE *origen, *destino;
7	char letra;
8	origen = fopen("origen.txt", "r");
9	destino = fopen("destino.txt", "w");
10	if (origen==NULL destino==NULL) {
11	printf("Problemas con los ficheros.\n");
12	exit(1);
13	}
14	letra=getc(origen);
15	while (feof(origen)==0) {
16	putc(letra, destino);
17	printf("%c", letra);
18	letra=getc(origen);
19	}
20	if (fclose(origen)!=0)
21	printf("Problemas al cerrar el fichero origen.txt\n");
22	if (fclose(destino)!=0)
23	printf("Problemas al cerrar el fichero destino.txt\n");
24	}

Como se ha visto el puntero FILE es la base de la escritura / lectura de archivos. Por eso se definen dos punteros FILE:

- el puntero 'origen' donde se almacenará la información sobre el fichero origen.txt y
- el puntero 'destino' donde se guardará el fichero destino.txt (el nombre del puntero no tiene por qué coincidir con el del fichero).

Introducción a la programación con C

El siguiente paso es abrir el fichero usando fopen, la diferencia es que ahora se deberá abrir para escritura, se usará el modo 'w' (crea el fichero o lo vacía si existe) porque se quiere crear un fichero.

Recordar que después de abrir un fichero hay que comprobar si la operación se ha realizado con éxito. En este caso, como es un ejemplo sencillo, se han comprobado ambos a la vez:

```
if (origen ==NULL || destino ==NULL)
```

Pero es más correcto hacerlo por separado, así se sabrá dónde se está produciendo el posible fallo.

Lectura del origen y escritura en destino- getc y putc

Como se puede observar en el ejemplo la lectura del fichero se hace igual como se hizo anteriormente. Para la escritura se usará la función **putc**:

```
int putc(int c, FILE *fichero);
```

Donde c contiene el carácter que se quiere escribir en el fichero y el puntero *fichero* es el fichero sobre el que se trabaja. De esta forma se irá escribiendo en el fichero *destino.txt* el contenido del fichero *origen.txt*.

Escritura de líneas - fputs

La función **fputs** trabaja junto con la función fgets:

```
int fputs(const char *cadena, FILE *fichero);
```

Ejercicio

Este programa lee un fichero y le suprime todas las vocales.
Es decir que siendo el fichero *origen.txt*:

“El alegre campesino
pasea por el campo
ajeno a que el toro
se acerca por detrás”

El fichero *destino.txt* sea:

```
I lgr cmpsn  
ps pr l cmp
```

Introducción a la programación con C

jn q l tr
s crc pr dtrás

Solución

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	
5	int main() {
6	FILE *origen, *destino;
7	char letra;
8	origen=fopen("origen.txt","r");
9	destino=fopen("destino.txt","w");
10	if (origen==NULL destino==NULL) {
11	printf("Problemas con los ficheros.\n");
12	exit(1);
13	}
14	letra=getc(origen);
15	while (feof(origen)==0) {
16	if (!strchr("AEIOUaeiou",letra)) putc(letra, destino);
17	letra=getc(origen);
18	}
19	if (fclose(origen)!=0)
20	printf("Problemas al cerrar el fichero origen.txt\n");
21	if (fclose(destino)!=0)
22	printf("Problemas al cerrar el fichero destino.txt\n");
23	}

Otras funciones para el manejo de ficheros

fread y fwrite

Las funciones vistas hasta ahora (getc, putc, fgets, fputs), son adecuadas para trabajar con caracteres (1 byte) y cadenas, pero ¿qué sucede cuando se quiere trabajar con otros tipos de datos?

Supón que se quieren almacenar variables de tipo int en un fichero. Como las funciones vistas hasta ahora sólo pueden operar con cadenas se deben convertir los valores a cadenas (con la función *itoa*), para recuperar luego estos valores se deben leer como cadenas y pasarlo a enteros (*atoi*).

Existe una solución mucho más fácil utilizando las funciones **fread** y **fwrite**. Estas funciones permiten tratar con datos de cualquier tipo, incluso con estructuras.

fwrite

Permite escribir en un fichero. Esta función tiene el siguiente formato:

```
fwrite(void *buffer, size_t size, size_t num, FILE *pfile);
```

- buffer - variable que contiene los datos que se van a escribir en el fichero.
- size - el tamaño del tipo de dato a escribir. Puede ser un int, un float, una estructura, etc. Para conocer su tamaño se usa el operador *sizeof*.
- num - el número de datos a escribir.
- pfile - El puntero al fichero sobre el que se trabaja.

Ejemplo: un programa de agenda que guarda el nombre, apellido y teléfono de cada persona.

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	
5	struct {
6	char nombre[20];

Introducción a la programación con C

7	char apellido[20];
8	char telefono[15];
9	} registro;
10	int main() {
11	FILE *fichero;
12	fichero = fopen("nombres.txt", "a");
13	do {
14	printf("Nombre: "); fflush(stdin);
15	gets(registro.nombre);
16	if (strcmp(registro.nombre,"")) {
17	printf("Apellido: "); fflush(stdin);
18	gets(registro.apellido);
19	printf("Teléfono: "); fflush(stdin);
20	gets(registro.telefono);
21	fwrite(®istro, sizeof(registro), 1, fichero);
22	}
23	}
24	while (strcmp(registro.nombre, " ")!=0);
25	fclose(fichero);
26	}

NOTA: El bucle termina cuando el 'nombre' se deja en blanco.

Este programa guarda los datos personales mediante fwrite, usando la estructura *registro*. Abrir el fichero en **modo 'a'** (append, añadir), para que los datos introducidos se añadan al final del fichero.

Una vez abierto se entra en un bucle *do-while* mediante el cual se introducen los datos. Los datos se van almacenando en la variable *registro* (que es una estructura). Una vez se tienen todos los datos de la persona, se meten en el fichero con fwrite:

- fwrite(Ristro, sizeof(registro), 1, fichero);
- Ristro - es la variable (en este caso una estructura) que contiene la información a meter al fichero.
- sizeof(registro) - lo utilizamos para saber cuál es el número de bytes a guardar, es decir el tamaño en bytes que ocupa la estructura.

Introducción a la programación con C

- 1 - indica que sólo se guardara un elemento. Cada vez que se recorre el bucle se guarda sólo un elemento.
- fichero - el puntero FILE al fichero donde vamos a escribir.

fread

La función *fread* se utiliza para sacar información de un fichero. Su formato es:

```
fread(void *buffer, size_t tamaño, size_t numero, FILE *pfichero);
```

Siendo *buffer* la variable donde se van a escribir los datos leídos del fichero *pfichero*.

El valor que devuelve la función indica el número de elementos de tamaño 'tamaño', que ha conseguido leer. Se le puede pedir a *fread* que lea 10 elementos (numero=10), pero si en el fichero sólo hay 6 elementos, *fread* devolverá el número 6.

Siguiendo con el ejemplo anterior, ahora se leerán los datos que se habían introducido en "nombres.txt".

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	
5	struct {
6	char nombre[20];
7	char apellido[20];
8	char telefono[15];
9	} registro;
10	
11	void main() {
12	FILE *fichero;
13	fichero = fopen("nombres.txt", "r");
14	while (!feof(fichero)) {
15	if (fread(Registro, sizeof(registro), 1, fichero)) {
16	printf("Nombre: %s\n", registro.nombre);

Introducción a la programación con C

```
17 printf("Apellido: %s\n", registro.apellido);
18 printf("Teléfono: %s\n", registro.telefono);
19 }
20 }
21 fclose(fichero );
22 }
```

Se abre el fichero *nombres.txt* en modo lectura. Con el bucle while se asegura que se recorre el fichero hasta el final (y que no nos pasamos).

La función fread lee un registro (numero=1) del tamaño de la estructura *registro*. Si realmente ha conseguido leer un registro la función devolverá un 1, en cuyo caso la condición del 'if' será verdadera y se imprimirá el registro en la pantalla. En caso de que no queden más registros en el fichero, fread devolverá 0 y no se mostrará nada en la pantalla.

Ejemplo

Escribir cinco registros en un archivo y leerlo posteriormente.

Solución

```
1 #include <stdio.h>
2
3 struct t_reg {
4     int num;
5     char cad[10];
6     char car;
7 };
8
9 int crear_archivo () {
10    FILE *fich;
11    int i, er_cod = 0;
12    struct t_reg r;
```

Introducción a la programación con C

```
13 if ((fich = fopen("fichreg.dat", "wb")) == NULL) {  
14     printf ("Error al abrir el archivo\n");  
15     er_cod = 1;  
16 }  
17 else {  
18     for (i = 0; i < 5; i + + ) {  
19         r.num = i;  
20         r.car ='a'+1;  
21         printf("Dé un nombre: ");  
22         gets(r.cad);  
23         fwrite(&r, sizeof(r), 1, fich);  
24     }  
25     fclose (fich);  
26 }  
27 return er_cod;  
28 }  
29  
30 int leer_archivo () {  
31     FILE *fich;  
32     struct t-reg r;  
33     int er_dev = 0;  
34     if ((fich = fopen("fichreg.dat", "rb")) == NULL) {  
35         printf ("Error abriendo el archivo para lectura\n");  
36         er_dev = 1.  
37     }  
38     else {  
39         fread (&r, sizeof(r), 1, fich);  
40         while (!feof(fich)) {  
41             printf ("%d: %s: %c\n" , r.num, r.cad, r.car);  
42             fread (&r, sizeof(r), 1, fich);  
43         }
```

Introducción a la programación con C

44	fclose (fich);
45	}
46	return er_dev;
47	}
48	int main(void) {
49	int error;
50	error = crear_archivo();
51	if (!error) leer_archivo();
52	}

EJEMPLOS: Archivos

1	#include <conio.h>
2	#include <stdio.h>
3	#include <string.h>
4	#include <io.h>
5	#include <stdlib.h>
6	#define NPacientes 4
7	void Imprimir(struct Paciente *P1);
8	void Elimina(struct Paciente *P1, int);
9	void Modifica(struct Paciente *P1);
10	int Buscar(struct Paciente *P1, int);
11	void Ingresar(struct Paciente *P1);
12	void Estadisticas(struct Paciente *P1);
13	void Guardar(struct Paciente *P1);
14	
15	struct Paciente
16	{
17	int REG;
18	int CEDULA;
19	char Nombre[12];
20	char Sexo[9];

Introducción a la programación con C

21	unsigned long TELEFONO;
22	char Seguro[9];
23	unsigned int CALLE;
24	unsigned int NUMERO;
25	char Ciudad[20];
26	};
27	Persona[NPacientes],*P1=Persona;
28	void main()
29	{
30	int SALIR=0, TECLA;
31	int SEARCH;
32	clrscr();
33	while(SALIR!=1)
34	{
35	printf("Clinica del Desamparado\n");
36	printf("1. Buscar En la Base De Datos:\n"),
37	printf("2. Estadisticas De La Empresa\n");
38	printf("3. Ingresar Nuevo Paciente:\n");
39	printf("4. Eliminar Paciente Paciente\n");
40	printf("5. Guardar Info En El DD:\n");
41	printf("99. Salir!: \n ");
42	printf("Escoja una opcion: ");
43	TECLA=getche();
44	switch(TECLA)
45	{
46	case 49:
47	int R=0;
48	clrscr();
49	printf("Ingrese la cedula de la persona:\n\n");
50	scanf("%d",&SEARCH);
51	R=Buscar(P1,SEARCH);

Introducción a la programación con C

52	if(R==1)
53	{
54	clrscr();
55	printf("\No sé ha encontrado la persona! :P\n");
56	printf("Se ingresará esta persona/n\n");getch();
57	Ingresar(P1);
58	}
59	Else
60	{
61	Imprimir(P1+R);
62	printf("PersonaEncontradaX:],Desea Modificarla?");
63	TECLA=getche();
64	if(TECLA=='s' TECLA=='S')
65	Modifica(P1+R);
66	}
67	break;
68	case 50:
69	Estadisticas(P1);
70	break;
71	case 51:
72	clrscr();
73	Ingresar(P1);
74	break;
75	case 52:
76	clrscr();
77	printf("Ingrese la cedula del paciente a borrar: ");
78	scanf("%d",&SEARCH);
79	Elimina(P1,&SEARCH);
80	break;
81	case 53:
82	clrscr();

Introducción a la programación con C

83	Guardar(P1);
84	break;
85	case 57:
86	TECLA=getche();
87	if (TECLA==57)
88	SALIR=1;
89	break;
90	}
91	}
92	
93	Buscar(struct Paciente *P1,int NUM) {
94	int i;
95	for(i=0;i<NPacientes;i++) {
96	if ((P1+i)->CEDULA == NUM)
97	return(i);break;
98	}
99	return(-1);
100	}
101	
102	void Imprimir(struct Paciente *P1) {
103	clrscr();
104	printf("\n\nCedula: ");
105	printf("%d",P1->CEDULA);
106	printf("Expedida en: ");
107	scanf("%s",P1->Ciudad);
108	printf("\n\nNombre");
109	scanf("%s",P1->Nombre);
110	printf("\n\nSexo: ");
111	scanf("%s",P1->Sexo);
112	printf("\n\nTelefono: ");
113	printf("%l",P1->TELEFONO);

Introducción a la programación con C

114	printf("\n\nDireccion:");
115	printf("\nCalle:");
116	printf("%d",P1->CALLE);
117	printf("\nNumero:");
118	printf("%d",P1->NUMERO);
119	printf("\n\nSeguro:");
120	scanf("%s",P1->Seguro);
121	}
122	
123	void Elimina(struct Paciente *P1,int NUM) {
124	int R;
125	R=Buscar(P1,NUM);
126	if (R== -1)
127	printf("Registro No Existente O Ya Borrado");
128	else {
129	(P1+R)->CEDULA=NULL;
130	strcpy((P1+R)->Nombre, NULL);
131	strcpy((P1+R)->Sexo, NULL);
132	strcpy((P1+R)->Ciudad, NULL);
133	strcpy((P1+R)->Seguro, NULL);
134	(P1+R)->TELEFONO=NULL;
135	(P1+R)->CALLE=NULL;
136	(P1+R)->NUMERO=NULL;
137	}
138	}
139	void Modifica(struct Paciente *P1) {
140	clrscr();
141	printf("Ingrese La Cedula: ");
142	scanf("%d",&P1->CEDULA);
143	printf("Ingrese El Nombre: ");
144	scanf("%s",P1->Nombre);

Introducción a la programación con C

145	printf("Ingrese El Sexo: ");
146	gets(P1->Sexo);
147	printf("Direccion:\n\n");
148	printf("Ingrese Calle: ");
149	scanf("%d",&P1->CALLE);
150	printf("Ingrese Numero: ");
151	scanf("%d",&P1->NUMERO);
152	printf("Tiene Seguro?\n");
153	gets(P1->Seguro);
154	}
155	void Ingresar(struct Paciente *P1) {
156	int i;
157	for(i=0;i<NPacientes;i++) {
158	if((P1+i)->REG==0) {
159	(P1+i)->REG=1;
160	clrscr();
161	printf("Ingrese La Cedula: ");
162	scanf("%d",&(P1+i)->CEDULA);
163	printf("Ingrese El Nombre: ");
164	scanf("%s",(P1+i)->Nombre);
165	printf("Ingrese El Sexo: ");
166	scanf("%s",(P1+i)->Sexo);
167	printf("Direccion:\n\n");
168	printf("Ingrese Calle: ");
169	scanf("%d",&(P1+i)->CALLE);
170	printf("Ingrese Numero: ");
171	scanf("%d",&(P1+i)->NUMERO);
172	printf("Ciudad: ");
173	scanf("%s",(P1+i)->Ciudad);
174	printf("Tiene Seguro?\n");
175	scanf("%s",(P1+i)->Seguro);

Introducción a la programación con C

176	break;
177	}
178	}
179	}
180	void Estadisticas(struct Paciente *P1) {}
181	void Guardar(struct Paciente *P1) {
182	FILE *ARCHIVO;
183	if((ARCHIVO=fopen("Data.vWv","wb"))==NULL)
184	exit(0);
185	Else
186	fwrite(P1,sizeof(Paciente),1,ARCHIVO);
187	fclose(ARCHIVO);
188	}

EJEMPLO 2:

1	#include <stdio.h>
2	
3	int main(int argc, char **argv)
4	{
5	FILE *fe, *fs;
6	unsigned char buffer[2048]; // Buffer de 2 Kbytes
7	int bytesLeidos;
8	if(argc != 3) {
9	printf("Usar:copia<archivo_origen><archivo_destino>\n");
10	return 1;
11	}
12	
13	// Abrir el archivo de entrada en lectura y binario
14	fe = fopen(argv[1], "rb");
15	if(!fe) {

Introducción a la programación con C

16	printf("%s no existe o no puede ser abierto.\n",
17	argv[1]);
18	return 1;
19	}
20	// Crear o sobreescribir el archivo de salida en binario
21	fs = fopen(argv[2], "wb");
22	if(!fs) {
23	printf("El archivo %s no puede ser creado.\n", argv[2]);
24	fclose(fe);
25	return 1;
26	}
27	// Bucle de copia:
28	while((bytesLeidos = fread(buffer, 1, 2048, fe)))
29	fwrite(buffer, 1, bytesLeidos, fs);
30	// Cerrar archivos:
31	fclose(fe);
32	fclose(fs);
33	return 0;
34	}

EJEMPLO 3

1	#include <stdio.h>
2	int main()
3	{
4	char nombre[10] = "datos.dat", linea[81];
5	FILE *archivo;
6	archivo = fopen(nombre, "r");
7	printf("Archivo: %s -> ", nombre)
8	if(archivo)
9	printf("existe (ABIERTO)\n");
10	else {

Introducción a la programación con C

11	printf("Error (NO ABIERTO)\n");
12	return 1;
13	}
14	printf("La primera linea del archivo: %s\n\n", nombre);
15	printf("%s\n", fgets(linea, 81, archivo));
16	if(!fclose(archivo))
17	printf("\nArchivo cerrado\n");
18	else {
19	printf("\nError: archivo NO CERRADO\n");
20	return 1;
21	}
22	return 0;
23	}

EJEMPLO 4

1	#include <stdio.h>
2	int main()
3	{
4	char nombre[11] = "datos2.dat";
5	FILE *archivo;
6	archivo = fopen(nombre, "w");
7	printf("Archivo: %s -> ", nombre);
8	if(archivo)
9	printf("creado (ABIERTO)\n");
10	else {
11	printf("Error (NO ABIERTO)\n");
12	return 1;
13	}
14	fputs("Esto es un ejemplo usando fputs\n", archivo);
15	if(!fclose(archivo))

16	printf("\nArchivo cerrado\n");
17	else {
18	printf("\nError: archivo NO CERRADO\n");
19	return 1;
20	}
21	return 0;
22	}

fseek

Cuando se lee un fichero, existe un 'puntero' que indica en qué lugar del mismo se encuentra el usuario. La función **fseek** permite al programador situarse en cualquier posición deseada de todo fichero abierto. Cada vez que se lean datos del fichero, este puntero se desplaza.

El formato de fseek es el siguiente:

```
fseek(FILE *pfichero, long desplazamiento, int modo);
```

```
fseek(nombre de apuntador, posición , origen);
```

Como siempre *pfichero* es un puntero de tipo FILE que apunta al fichero con el que se quiere trabajar, *desplazamiento* son las posiciones (o bytes) que se quiere desplazar el puntero. Este desplazamiento puede ser de tres tipos dependiendo del valor de *modo*:

SEEK_SET	El puntero se desplaza desde el principio del fichero.
SEEK_CUR	El puntero se desplaza desde la posición actual del fichero.
SEEK_END	El puntero se desplaza desde el final del fichero.

Estas tres constantes están definidas en el fichero <stdio.h>. Como curiosidad se indican a continuación sus definiciones:

```
#define SEEK_SET 0  
#define SEEK_CUR 1
```

Introducción a la programación con C

```
#define SEEK_END      2
```

Nota: es posible que los valores cambien de un compilador a otro.

La función devuelve un valor distinto de 0 si existen errores, si todo ha ido bien el valor devuelto es un 0.

El funcionamiento de fseek se puede mostrar mediante el siguiente ejemplo, un programa que lee la letra que hay en la posición que especifica el usuario.

1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <conio.h>
4	void main()
5	{
6	FILE *fichero;
7	long posicion;
8	int resultado;
9	fichero = fopen("origen.txt", "r");
10	printf("¿Qué posición quieres leer? ");
11	fflush(stdout);
12	scanf("%d", &posición);
13	resultado = fseek(fichero, posición, SEEK_SET);
14	if (!resultado)
15	printf("%c está en la posición %d\n", fgetc(fichero), posición);
16	Else
17	printf("Problemas posicionando el cursor.\n");
18	fclose(fichero);
19	}

ftell

Esta función es complementaria a fseek, devuelve la posición actual dentro del fichero.

Introducción a la programación con C

Su formato es el siguiente:

```
long ftell(FILE *pfichero);
```

El valor que da ftell puede ser usado por fseek para volver a la posición actual.

fprintf y fscanf

Estas dos funciones trabajan igual que sus equivalentes printf y scanf. La única diferencia es que se puede especificar el fichero sobre el que opera (si se desea puede ser la pantalla para fprintf o el teclado para fscanf).

Los formatos de estas dos funciones son:

```
int fprintf(FILE *pfichero, const char *formato, ...);
int fscanf (FILE *pfichero, const char *formato, ...);
```

Ejercicios Propuestos

1. Escribir un programa que cuente el número de letras minúsculas, de letras mayúsculas y de cifras de un fichero. Nota: en la biblioteca ctype hay funciones que se pueden usar para este ejercicio.
2. Escribir un programa que copie el contenido de un fichero en otro.
3. Escribir una función que copie una línea de un flujo en otro. La función tendrá como parámetros el flujo en entrada, el flujo de salida y un unsigned int que indica cual es el número de la línea que se va a copiar.
4. Escribir un programa con la opción de encriptar y de desencriptar un fichero de texto. La encriptación (codificación) consiste en que dado un fichero de texto de entrada genere otro fichero de salida de extensión .COD donde el texto estará codificado (encriptado). Esta codificación consiste reemplazar cada carácter por el tercero siguiente (ej. 'a' → 'd'). Si el carácter resultante es no imprimible éste no se cambia. La opción de desencriptado consiste en leer un fichero .COD y recuperar la información original.
5. Escribir un programa para procesar información sobre los clientes de una agencia matrimonial. El programa debe crear un fichero de texto llamado "PERSONAS.DAT" en el que se guarde la información de un número indeterminado de personas. La

Introducción a la programación con C

información que se guardar 'a por cada persona será: 18 TEMA 1. GESTION BASICA DE FICHEROS ' Nombre: De 1 a 30 caracteres. Edad unsigned int. Sexo char (M/F). Arte char (S/N). Deporte char (S/N). Libros char (S/N). Música char (S/N). La información correspondiente a cada persona se leerá del teclado. El proceso finalizar 'a cuando se teclee un campo Nombre que este vacío.

- 6.** Ampliar el programa que procesa clientes de una agencia matrimonial para que tome los datos de todos los candidatos a estudiar del fichero PERSONAS.DAT del ejercicio anterior, lea el cliente del teclado y finalmente genere como resultado un listado por pantalla con los nombres de los candidatos aceptados y un fichero llamado ACEPTADOS.DAT con toda la información correspondiente a los candidatos aceptados. Una persona del fichero PERSONAS.DAT se considerara aceptable como candidato si tiene diferente sexo y por lo menos tres aficiones comunes con respecto al aspirante introducido por pantalla. (El programa debe ser capaz de trabajar con cualquier número de personas en el fichero PERSONAS.DAT). 7.
- 7.** Codifique un programa que cree un fichero para contener los datos relativos a los artículos de un almacén. Para cada artículo habrá de guardar la siguiente información: Código del artículo (Numérico) Nombre del artículo (Cadena de caracteres) Existencias actuales (Numérico) Precio (Numérico) Se deberán pedir datos de cada artículo por teclado hasta que como código se teclee 0. El fichero se habrá de crear ordenado por el código del artículo.
- 8.** Escriba un programa que dados dos ficheros generados por el programa anterior y ordenados genere un tercer fichero que sea el resultado de mezclar de forma ordenada los dos primeros.
- 9.** Escriba un programa que tome como entrada el fichero del ejercicio 4 y una condición sobre los campos Existencias actuales o Precio. La condición podrá ser: [<, <=, >, >=, <>] Este programa debe generar como salida un fichero llamado salida.dat que contenga todos aquellos artículos para los que se cumple la condición de entrada.
- 10.** Escriba un programa que tenga como entrada un fichero que contenga un texto y dé como resultado una estadística de las palabras que hay de cada longitud, así como de los caracteres especiales (", ", ., :, ;).

Conclusiones

El abarcar conceptos fundamentales de la programación en un lenguaje tan robusto en sus aplicaciones, y tan complejo debido a su cercanía de comunicación con la maquina (computadora), es un trabajo altamente importante para las épocas actuales en donde la revolución tecnológica sigue avanzando a ritmos inigualables. Con

Introducción a la programación con C

ello, debemos ser conscientes que este lenguaje dio entrada a muchas mas funcionalidades como el manejo de memoria detallado, potencia y rapidez en compilación, multiplataforma y constante actualización, entre otras. Un dicho bastante común en la comunidad general de programadores dice que “quien programa en C, generalmente programa en lo que sea”, y como muestra de ello, el sistema operativo de Windows en su versión 10 está escrito mayormente en C, en compañía de otros lenguajes emergentes durante los últimos años.

De lo anterior, podemos entonces mencionar que el desarrollo de todo el texto concluye todos los conocimientos básicos del lenguaje de programación C, abarcando conceptos generales de la programación y la informática, de la mano con el paradigma de la programación funcional e imperativa. Con ello, es una gran ventaja el asegurar que el texto esté compuesto para un método de aprendizaje eficaz en donde pueden ser participes desde estudiantes universitarios hasta aquellos pertenecientes a la media secundaria. Cada capítulo se logró explicar de una manera simple y concreta proporcionando todo lo necesario para una amplia comprensión. Se resalta la forma en como se compone cada idea explicada, un apartado con teoría que luego se apoya de ejemplos de distinta dificultad ilustrados mediante la codificación en la sintaxis del lenguaje e imágenes descriptivas. A su vez se provee de mas ejercicios para fomentar un refuerzo en los conocimientos adquiridos.

Por ultimo, los conceptos mencionados fueron considerados según la experiencia del autor, persiguiendo el poder desarrollar bases solidas de conocimientos en programación.

Bibliografía

Los tipos abstractos de datos. Estructuras de datos y algoritmos 03/04.

<<http://webdiis.unizar.es/~elvira/eda/material0304/TADespec/TAD.pdf>>
(25 Feb 2013)

Universidad de Costa Rica. "Tipos abstractos de datos y programación".

Di Mare, Adolfo: *Tipos abstractos de datos y programación por objetos. Reporte técnico PIBDC-03-91*, proyecto 326-89-019. Escuela de Ciencias de la Computación e Informática, UCR 1991.
(25 Feb 2013)

Luis Lastra Cid. "Estructuras de datos y tipos abstractos de datos ".
<Instituto Profesional Virginio Gómez>
(25 Feb 2013)

Jorge A. Villalobos. "Diseño y manejo de Estructuras de Datos en C"
Editorial McGraw-Hill, Enero 1996.
(27 Mar 2013)

Villalobos S, Jorge A; Castillo, María Fernanda. Introducción a las estructuras de datos: aprendizaje activo basado en casos. Bogotá D.C: Pearson Educación de Colombia, 2008. (27 Mar 2013)

Luis Joyanes Aguilar Ignacio Zahonero Martínez
Estructura de datos algoritmos Fecha publicación: 2007
Editorial: McGraw-Hill Interamericana Colección: 1^a Edición / 512 págs. / Rústica / Castellano / Libro ISBN13:9788448156312

Villalobos S, Jorge A. "Diseño y manejo de Estructuras de Datos en C" Editorial McGraw-Hill, Enero 1996.
(27 Mar 2013)

Villalobos S, Jorge A; Castillo, María Fernanda. Introducción a las estructuras de datos:
aprendizaje activo basado en casos. Bogotá D.C : Pearson Educación de Colombia,
2008.
(27 Mar 2013)

Instituto Específico Formación Profesional Superior "BIDASOA. PROGRAMACIÓN EN C++ Y VISUAL BASIC, 2009. M. S. Zigor Aldazabal.

Introducción a la programación con C

(2009)

Aparicio Gil et al. TEMA 2: Lenguajes de programación, pag 8;
Lenguajes de bajo nivel – inconvenientes.

(s.f.)

García de Jalón J., Ignacio Rodríguez J. et al. Universidad de
Navarra, Curso completo de visual Basic 6.0.

(s.f.)

Cita -> 1ª parte: Metodología de Programación
< <http://mimosa.pntic.mec.es/~flarrosa/pseudoco.pdf> >
(s.f.)

Machaca M. A. Introducción a l programación en visual Basic .NET
2008
(2014)

Facultad de Ciencias de la computación, Benemérita universidad
Autónoma de Puebla. Algoritmos de ordenación
(s.f.)

Tema 1. Gestión básica de ficheros
< <http://www.lcc.uma.es/~lopez/lp2/apuntes/01-ficheros/ficheros.pdf> >
(s.f.)

Universidad de Malaga, departamento de lenguajes y CC de la
computación . Programación II Relación de Ejercicios Sonido e
Imagen
(s.f.)

E.T.S.I. Técnicos en Informática de Sistemas Laboratorio de
Programación. 1ºB
< <http://www.lcc.uma.es/~afdez/apuntes/laboratorio/practica5/practica5.pdf> >
(s.f.)

9. GESTION BÁSICA DE FICHEROS.
< http://winuxnet.com/ArchivosPDF/Programacion/Apuntes_fichero.pdf >
(s.f.)

Algoritmos
< <https://www.buenastareas.com/ensayos/Algoritmos/7801158.html> >
(2013)

Introducción a la programación con C

Ing. sistemas programación

< <https://www.buenastareas.com/ensayos/Ing-Sistemas-Programacion/2101489.html> >
(2011)

Historia de los lenguajes de programación

< <http://informatix-universe.blogspot.com/2014/11/historia-de-los-lenguajes-de.html> >
(2014)

Proyecto sm1

< proyecto_sm1-febrero_2013_proyectosub1.blogspot.com >
(9 de febrero, 2013)

Trejos Buriticá, Omar Ivan. La Esencia de la Lógica de Programación – Básico. Editorial Papiro, 1999.
(1999)

j. Antonio Lemos y Eduardo Victoria Z. Informática 1 (algoritmos con Java), Universidad Autónoma de Occidente , Facultad de Ingenierías.
(2004)

Historia del lenguaje C.

< <http://mistica.freeservers.com/c.htm> >
(s.f.)

Guillermo Carmona Quintana. Resolución de Problemas por computadora

< <http://docplayer.es/277346-Resolucion-de-problemas-por-computadoras.html> >
(2015)

FELIPE RESTREPO CALLE . PROGRMACIÓN II INGRENIERÍA FÍSICA PRIMERA PARTE, UNIVERSIDAD TECNOLÓGICA DE PEREIRA.
(2005)

Estructura de Datos Antología

< <https://es.scribd.com/doc/2873588/Estructura-de-datos-Antologia> >
(2008)

Jose Luis. Funciones en programación

< <http://jolextic0.blogspot.com/2011/03/funciones-en-programacion.html> >
(2011)

conceptos avanzados sobre el C

Introducción a la programación con C

< <https://hosting.miarroba.com/indice.php> >

(s.f.)

Referencias

(Algoritmos, 2013). Algoritmos.

(Historia de los lenguajes de programación, 2014, párrafo 3). Historia de los lenguajes de programación.

(Ing. de sistemas programación, 2011). Ing. de sistemas programación

(Proyecto sm1 ,2013, párrafo 47). Proyecto sm1

(Trejos, 1999). La Esencia de la Lógica de Programación – Básico

(Lemos y Victoria, 2004). Informática 1 (algoritmos con Java), Universidad Autónoma de Occidente , Facultad de Ingenierías.

(Historia del leguaje C, s.f.). Historia del lenguaje C.

(Quintana, 2015). Resolución de Problemas por computadora

(Calle, 2005). PROGRMACIÓN II INGRENIERÍA FÍSICA PRIMERA PARTE, FELIPE RESTREPO CALLE, UNIVERSIDAD TECNOLÓGICA DE PEREIRA. 2005

(Estructura de datos Antología, 2008).Estructura de Datos Antología

(Facultad de Ciencias de la computación, Benemérita universidad Autónoma de Puebla [BUAP], s.f.). Facultad de Ciencias de la computación, Benemérita universidad Autónoma de Puebla. Algoritmos de ordenación

(Luis, 2011). Funciones en programación.

(conceptos avanzados sobre el C, s.f.). conceptos avanzados sobre C

(Zigor, 2009, ...). PROGRAMACIÓN EN C++ Y VISUAL BASIC, 2009 CITA

(Gil et al, s.f.).TEMA 2: Lenguajes de programación

Introducción a la programación con C

(de Jalón et al, s.f.).Universidad de Navarra, Curso completo de visual Basic 6.0

(1^a parte: Metodología de Programación, s.f.) .1^a parte: Metodología de Programación

(Machaca M. (2014, Cap 9-12)).Introducción a l programación en visual Basic .NET 2008