Programación básica

MANUEL JOSUE ESCOBAR CRISTIANI

RED TERCER MILENIO

PROGRAMACIÓN BÁSICA

PROGRAMACIÓN BÁSICA

MANUEL JOSUE ESCOBAR CRISTIANI

RED TERCER MILENIO

AVISO LEGAL

Datos para catalogación bibliográfica

Manuel Josué Escobar Cristiani

Programación básica

ISBN 978-607-733-168-1

Primera edición: 2012

DIRECTORIO

ÍNDICE

INTRODUCCIÓN

Desde 1978, año en que nace el lenguaje de programación “C” el mundo de las computadoras ha cambiado enormemente, las grandes computadoras de entonces tienen menos recursos que un pequeño equipo personal moderno,

dentro del gran desarrollo de los sistemas de cómputo podemos mencionar que una gran computadora de hace más de 30 años, ocupaba una habitación de más de 30 metros cuadrados y contaba con las siguientes características: memoria principal de 64,000 Bytes, sin gigas, megas o kilos, su único dispositivo de entrada era por medio de tarjetas de cartón perforadas, y todos los resultados los arrojaba impresos en papel continuo en una impresora del tamaño de una lavadora moderna, su pequeño teclado y su monitor monocromático servían para que los grandes gurús de programación pudieran encender y apagar dicha computadora, sin embargo en estos 32 años el lenguaje de programación “C” solo ha sufrido cambios muy modestos.

En 1983, el American National Standars Institute (ANSI) estableció un comité cuyo propósito era el desarrollo de un estándar de lenguaje “C” que fuera totalmente independiente al equipo de cómputo en el que se utilizara, naciendo de esta forma el estándar ANSI del lenguaje “C”.

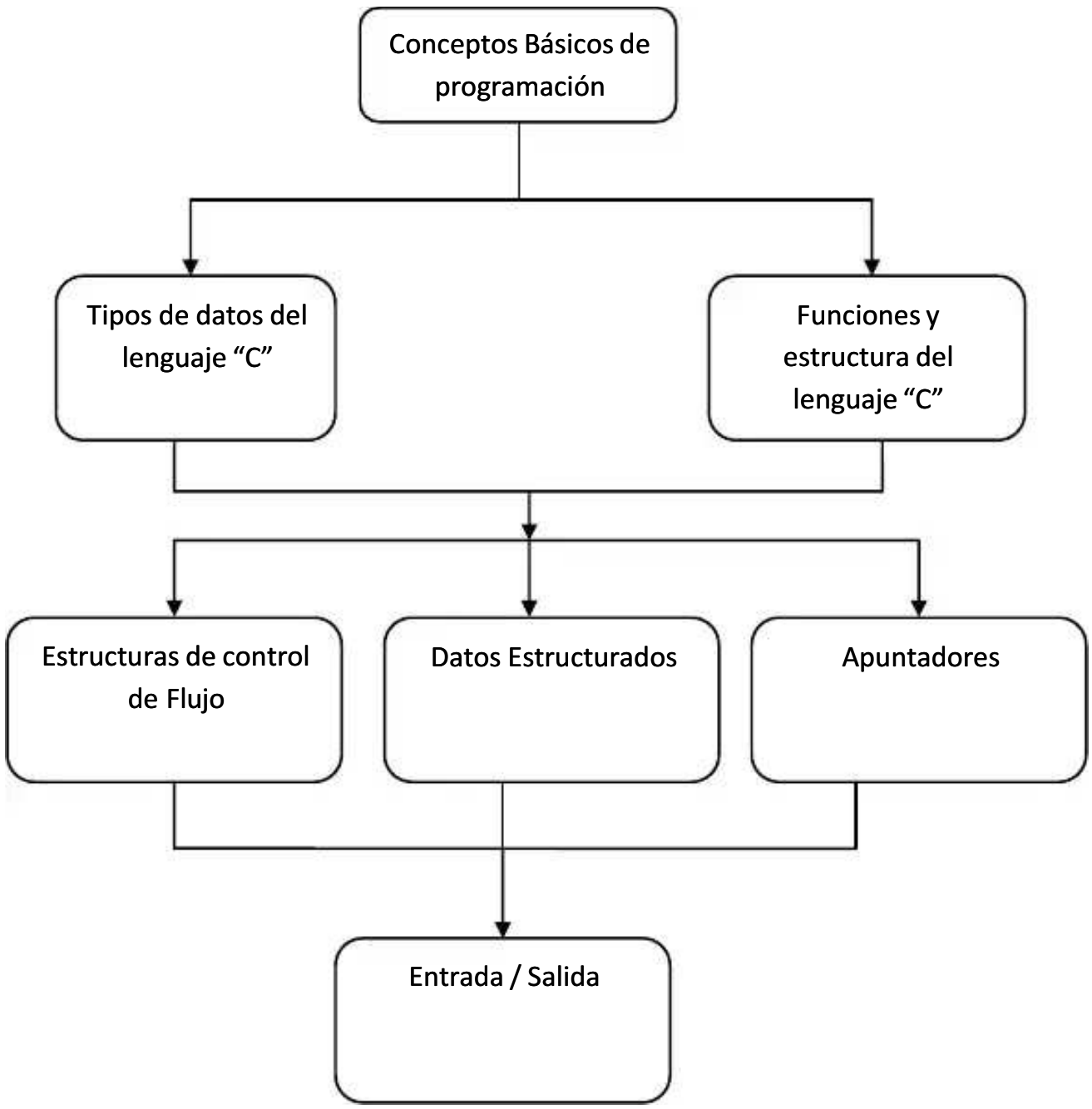
El objetivo principal de este libro es enseñarle al alumno el lenguaje de programación “C”; para lograr este objetivo es muy importante, que durante el

curso, el alumno tenga acceso a un equipo con el compilador “C” y así aprender mediante la programación constante de ejemplos y proyectos.

Es por esta razón, que desde el capítulo 1 presentamos nuestro primer programa en “C”, para comprender mejor lo explicado en este libro.

Espero que al finalizar el libro el estudiante sea capaz de resolver problemas de programación mediante la utilización de este lenguaje de programación.

MAPA CONCEPTUAL



UNIDAD 1

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

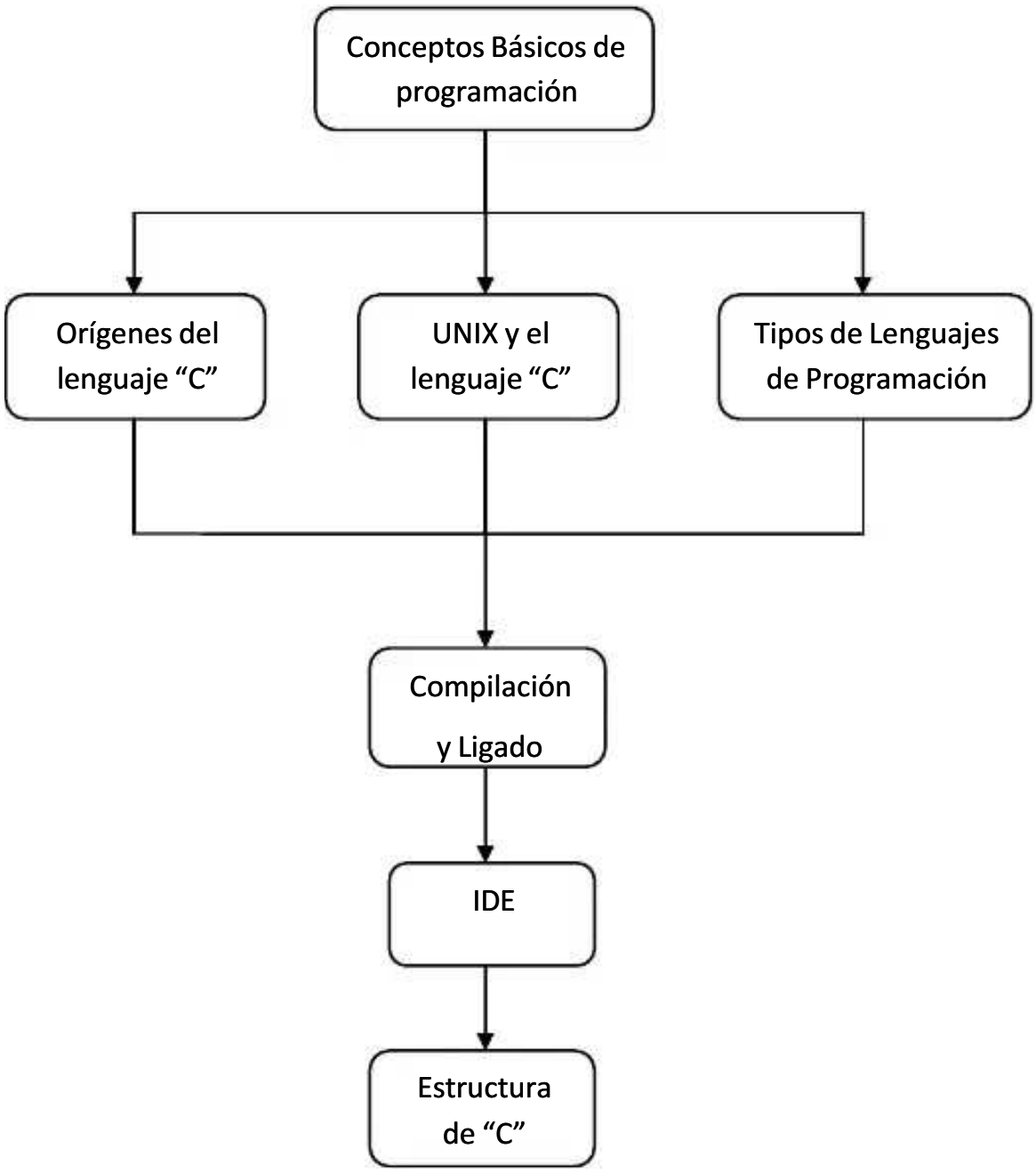
OBJETIVO:

Que el estudiante aprenda cómo nació el lenguaje de programación “C” y su importancia en el desarrollo de los sistemas de cómputo, así como algunos conceptos básicos, pero en extremo importantes, utilizados en la programación de sistemas

---

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

El lenguaje C fue desarrollado hace 32 años, durante más de 3 décadas a conservado su esencia sin sufrir grandes cambios, en este capítulo se mostrará la estructura de un programa escrito en “C”, como logramos que las instrucciones escritas por los programadores lleguen a ser entendibles por las computadora, es decir, convertir el código a lenguaje de unos y ceros. También escribiremos nuestro primer programa en “C” para poder condensar lo aprendido en un programa sencillo.

1.1 ORÍGENES DEL LENGUAJE "C"

"C" es un lenguaje de programación de propósito general y de tecnología abierta, es decir no depende de la máquina ni del sistema operativo con el que opere. Gran parte de los fundamentos de "C" provienen del lenguaje BCPL (Basic Combined Programming Languaje o Lenguaje de Programación Básico Combinado) que fué desarrollado por Martin Richards en la Universidad de Cambridge en 1966.

"C" proporciona construcciones fundamentales de control de flujo, una variedad de tipos de datos principalmente: caracteres, enteros y números de punto flotante, también integra el uso de funciones con paso de parámetros.

"C" se puede considerar un lenguaje de programación de "medio nivel", con esto no queremos decir que es menos potente o menos importante que otros lenguajes conocidos de "alto nivel".

"C" tiene una estructura y funciones que lo acercan a los lenguajes ensambladores, ya que es una representación simbólica del código máquina, no es muy grande, pero su funcionalidad es similar a los lenguajes de "alto nivel", combinando elementos propios de estos lenguajes.

Durante varios años la definición de "C" fue el manual de referencia de la primera edición de "El lenguaje de programación C", fue hasta 1983 cuando el American National Standards Institute (ANSI) estableció un comité para proporcionar una definición de C, resultando, así, el estándar ANSI C[1], este estándar internacional tiene el propósito de garantizar la portabilidad de programas desarrollados con el lenguaje de programación C a través de una gran variedad de sistemas de procesamiento de datos, es decir hacerlo completamente independiente al equipo de cómputo y al sistema operativo en

---

[1] ISO/IEC 9899:1999 Programming languajes –C

http://webstore.ansi.org/RecordDetail.aspx?sku=ISO/IEC+9899:1999

que sea utilizado, y está pensado para implementadores y programadores de sistemas.

1.2 UNIX Y EL LENGUAJE "C"

Como se mencionó en la introducción, este lenguaje de programación nació hace 32 años, es decir en 1978, C se desarrolló, originalmente, para el sistema operativo UNIX por Dennis Ritchie, quien fue el creador de este lenguaje de programación, lo implantó, en un principio, en una computadora DEC (Digital Equipment Corporation) PDP-1, es importante mencionar que el sistema operativo, el compilador C y los programas de aplicación de UNIX están escritos en C.

C en una primera instancia fue utilizado para la programación de Sistemas (programas que hacen que las computadoras sean capaces de realizar algún trabajo útil), esto se debió a que un programa desarrollado en "C" puede ser tan veloz al ejecutarse como un programa desarrollado en lenguaje ensamblador, aunque es más sencilla la programación en "C" en comparación con la programación en lenguaje ensamblador. Su velocidad de ejecución es una de las principales características de este lenguaje.

1.3 LENGUAJES DE MÁQUINA, LENGUAJES ENSAMBLADORES Y LENGUAJES DE ALTO NIVEL

Los lenguajes de programación, al igual que nuestros lenguajes habituales como pueden ser: el español, el inglés, el italiano, etc., deben poseer una estructura o sintaxis y un significado o semántica.

Por ejemplo, la lengua española integra una serie de reglas para poder combinar palabras y con esto formar oraciones o frases que puedan ser entendibles por cualquier persona que conozca dicha lengua. De igual manera

los lenguajes de computadoras establecen reglas muy claras para su correcto uso.

Resumiendo: “un lenguaje de programación es un conjunto de reglas, símbolos y palabras especiales que permiten construir un programa”[2]

Existen centenares de lenguajes de programación para computadoras, cada uno de los cuáles puede tener diferentes versiones, así como ventajas y desventajas. Con base en la cantidad de instrucciones requeridas por cada uno de estos lenguajes para realizar una tarea específica, se pueden clasificar en: bajo nivel y alto nivel, además los lenguajes de bajo nivel están más cercanos a las máquinas y son difíciles de entender por los programadores, en cambio, los lenguajes de alto nivel son cercanos y entendibles para la gran mayoría de los programadores.

Como lenguajes de bajo nivel podemos citar:

- lenguajes de máquina
- lenguajes ensambladores.

Lenguaje de Máquina

Un lenguaje de máquina es el único lenguaje que, realmente puede entender una computadora, es un conjunto de reglas sintácticas, escritos exclusivamente con un conjunto de unos y ceros, por ejemplo:

1000110010010001

1100011101010110

1100110110101101

Estos lenguajes son muy difíciles de programar y de entender -solamente programadores expertos y muy capacitados podrían entenderlos- a la vez que

---

2 Metodología de la Programación Diagramas de flujo, algoritmos algoritmos y programación estructurada. Luis Joyas Aguilar. Mc Graw Hill

son muy largos, como fueron los primeros programas en aparecer se les conoce como *“lenguaje de primera generación”*.

Estos lenguajes son desarrollados para ser utilizados en una sola máquina y en un solo procesador, aunque son entendibles para la máquina en que fueron desarrollados es muy difícil ser entendido por los programadores.

Lenguaje ensamblador

Para ser más sencillos de entender por los programadores se desarrollaron los lenguajes ensambladores, son más fáciles de utilizar que el lenguaje de máquina, y son únicos para un procesador en particular (Z80, 8080, Pentium IV, etc.), utilizan símbolos para interpretar las instrucciones en lugar de largas cadenas de 1 y 0

Por ejemplo:

```
LR  5,0
M 4,=F’4’
4,7
```

No entraremos en detalle de explicar las instrucciones de este tipo de lenguajes, ya que solo aplicaría para un procesador en particular, y se sale de los objetivos de este libro.

Para muchos especialistas en la historia del software constituyen la “*segunda generación”* de los lenguajes de programación

Lenguajes de Alto Nivel

Como podemos entender, existe el interés por el desarrollo de lenguajes de programación cada vez más sencillos, con instrucciones que puedan “leerse” con facilidad y de ser posible no dependan de la máquina ni del procesador utilizados, que sirvan para el desarrollo de algoritmos para ejecutarse en la mayoría de los equipos de cómputo.

Una sola instrucción en un lenguaje de alto nivel supone varias instrucciones en lenguaje de máquina.

Por ejemplo: Z= X + Y / 2 + C*SEN(N) puede equivaler, tal vez, a centenares de instrucciones en lenguaje de máquina.

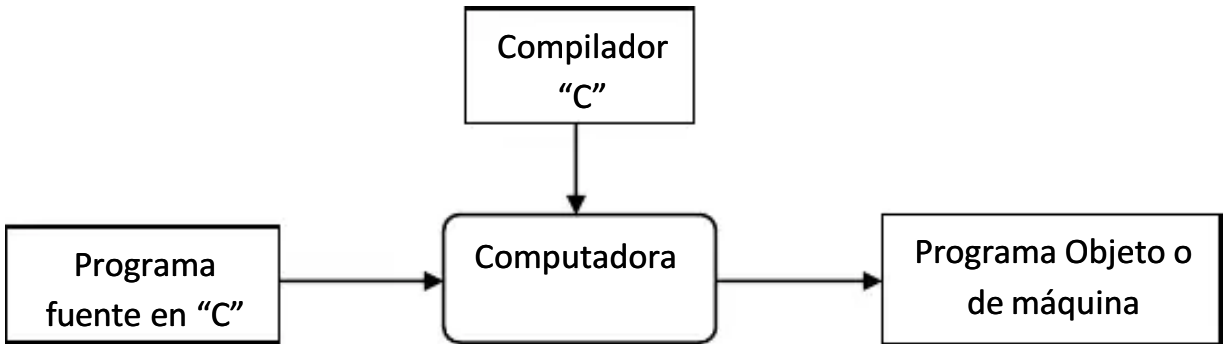


Figura 1) Proceso de creación del código de máquina a partir de un lenguaje de alto nivel

Se conocen como *“Lenguajes de Tercera Generación”*

Un Lenguaje de Alto nivel requiere de procesos intermedios (Compilación y Ligado) para poder ser entendido por una computadora, es decir para generar el código de máquina.

Actividad de aprendizaje

El estudiante deberá investigar como compilar y ligar un programa desarrollado en “C” en los equipos con que cuente la escuela, así mismo debe

comparar las instrucciones de compilación y ligado por lo menos con otros 3 lenguajes de alto nivel.

1.4 Compilación y Ligado

Como se mencionó en el tema anterior para que un programa de alto nivel pueda ser ejecutado en una computadora, es preciso realizar los procesos de compilación y ligado de dicho programa.

Compilación: Proceso de verificar las instrucciones escritas en el lenguaje de alto nivel para garantizar que no tenga errores de sintaxis, es decir que no tenga un formato no válido. Si al compilar el programa la computadora regresa errores el programador debe revisarlos para tomar las correcciones necesarias.

Un compilador lee el programa en su totalidad y lo convierte en código objeto, llamado también código binario o de máquina.

Algunos compiladores detienen su ejecución al encontrar el primer error, otros en cambio analizan el programa en su totalidad y entregan una lista de todos los errores encontrados, de esta manera se optimiza el tiempo de los programadores.

En ocasiones aunque la compilación sea exitosa, los programas no muestran los resultados deseados, esto se debe a errores en la lógica de la programación, un ejemplo muy claro es la de dividir entre cero una cierta cantidad, lo que ocasiona un error al momento de ejecutar el programa ya compilado, durante el proceso de compilación, si la instrucción está bien escrita, este error no puede ser señalado.

Una vez que un programa de alto nivel se ha compilado es necesario utilizar un proceso que lo “una” a todas las funciones del lenguaje o a otros programas previamente desarrollados, a este proceso se le conoce como Ligado.

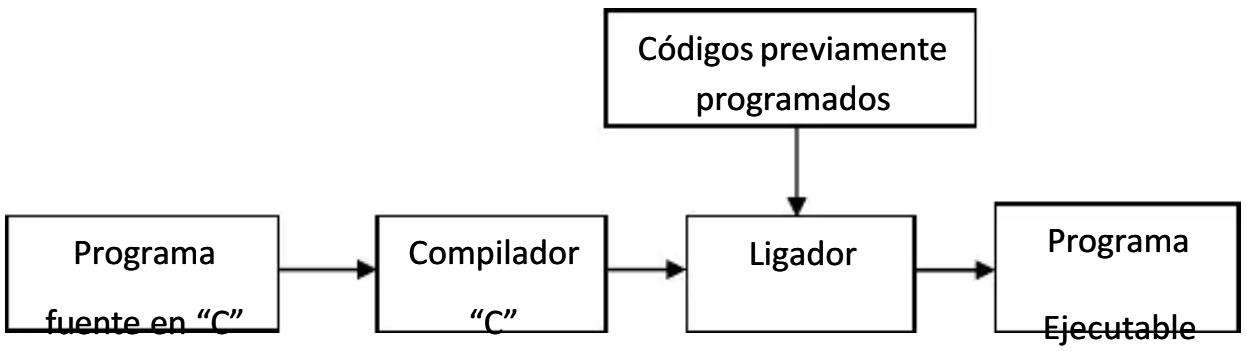


Figura 2) Proceso de creación del programa ejecutable a partir de un lenguaje de alto nivel

Recordemos que el código fuente es el texto de un programa escrito por los programadores y el programa ejecutable es el código generado por la compilación y el ligado y es una secuencia de instrucciones escritas con unos y ceros.

Cada vez que se realicen cambios en el programa fuente –escrito en lenguaje de alto nivel- será necesario volver a compilar y a ligar dicho programa para obtener así el nuevo código ejecutable.

1.5 El entorno integrado de desarrollo (IDE)

Un entorno de desarrollo integrado, IDE (por sus siglas en inglés: **I**ntegrated **D**evelopment **E**nvironment), es una aplicación informática compuesta por un conjunto de herramientas de programación, que permitan realizar todas las fases de puesta a punto de un programa, esta aplicación debe incluir: un editor de código, un compilador, un ligador, una interfaz gráfica, etc. Tiene la finalidad de hacer el trabajo de los programadores mucho más sencillo englobando las herramientas de desarrollo dentro del ambiente de cómputo del programador, además de permitir la programación de aplicaciones de una manera rápida y sencilla.

Puede dedicarse exclusivamente a un sólo lenguaje de programación o bien, puede utilizarse para varios.

Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Por ejemplo El lenguaje Visual Basic, puede ser usado dentro de otras aplicaciones de Microsoft Office, con esto se logra escribir sentencias Visual Basic en forma de macros para Microsoft Word, o Microsoft Excel.

Los IDE proveen un marco de trabajo amigable para los programadores en la mayoría de los lenguajes de programación tales como C, C++, Python, Java, C#, Delphi, Visual Basic, etc.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación.

Como ejemplos de IDE podemos citar:

- Visual Basic, que es un entorno visual e interactivo.
- Eclipse, al que mediante plugins se le puede añadir soporte de lenguajes adicionales.
- Dev-C++, es un Entorno Integrado de Desarrollo para el lenguaje de programación C/C++
- IDE PHP

Principalmente un IDE debe incluir los siguientes componentes:

- Editor de texto.
- Compilador.
- Intérprete, dependiendo del lenguaje de programación utilizado los intérpretes se utilizan en lugar de los compiladores.
- Diversas herramientas de automatización.
- Debbuger (depurador).
- Posibilidad de ofrecer un sistema de control de versiones.

- Interfaces gráficas de usuario.

Actividad de aprendizaje

El estudiante deberá investigar cual es el Entorno Integrado de Desarrollo que será utilizado durante el curso, investigará cuáles son sus componentes principales y cómo pueden utilizar dichos componentes.

1.6 Estructura de un Programa en "C"

Un programa escrito en lenguaje C está compuesto por una o más funciones. Como sabemos, una función es un programa que realizará una tarea determinada, por ejemplo, la función printf sirve para imprimir datos en la salida estándar de C.

En el lenguaje de programación C existe una función que debe estar presente en todos los programas escritos en este lenguaje, sirve para marcar el inicio y el fin de la ejecución de cualquier programa realizado; esta es la función principal, la primera que se ejecuta, a partir de ella se inician todas las instrucciones que deban ser ejecutadas, es la función "main". Su sintaxis es:

```
main()
        {
                <conjunto_de_instrucciones>
        }
```

La función inicial de C "main" contiene el conjunto de instrucciones de un programa, dichas instrucciones se ubican dentro de los caracteres abrir llave ({) y cerrar llave (}).

Recordemos: los picoparéntesis < > se utilizan para colocar dentro de ellos una o más instrucciones y, en este caso, no deben ser escritas como parte

del programa C, solo las hemos utilizado como una notación, en cambio las llaves { } indican el principio y el fin de una función escrita en C y deben ser escritas en el programa.

Los paréntesis "()" escritos después de “main” sirven para indicar que el identificador main es una función, y que no recibe argumentos, es importante comprender que main no es una palabra reservada de C.

Dentro de las llaves podemos escribir todo nuestro programa, realizar llamadas a otras funciones, ya sean escritas por el mismo programador o almacenadas dentro de las librerías del lenguaje de programación C.

Si se van a utilizar librerías del lenguaje de programación, antes del nombre de la función deben indicarse las librerías a utilizarse, por ejemplo:

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

Esta instrucción le indica al compilador C que incluya información de la biblioteca estándar de entrada/salida (estándar input/output), es decir se incluirá un conjunto de funciones que proporcionan entrada y salida de información al programa o función que lo requiera, esta biblioteca ha sido desarrollada como parte del lenguaje de programación C.

1.7 Mi primer programa en “C”

Para comprender mejor la estructura de un programa en C, y teniendo en cuanta que la mejor forma de aprender un nuevo lenguaje de programación es escribiendo programas con él, escribiremos nuestro primer programa en C, el objetivo de este programa es muy sencillo, imprimir las palabras:

Buenos dias amigos

Para lograr este objetivo se debe de escribir el siguiente programa en “C”:

```
#include <stdio.h>
main()
{
printf(“Buenos días amigos\n”);
}
```

Este sencillo programa nos sirve para comprender los puntos señalados en el tema 1.6, donde aprendimos la estructura básica de un programa en “C” y nos prepara para entender mejor los siguientes capítulos donde aprenderemos las principales funciones existentes en el lenguaje “C”.

Para comprender mejor este programa daremos las siguientes explicaciones:

Un programa en C está compuesto de funciones y variables.

Una función contiene proposiciones, las cuáles especifican todas las operaciones de cálculo que deben ser realizadas, estas son similares a las “subrutinas” o “procedimientos” utilizados en otros lenguajes de programación.

Las variables almacenan los valores utilizados durante los cálculos.

Una función puede tener el nombre que el programador desee, pero “main” es la función principal de un programa en C, cualquier programa escrito en C comienza a ejecutarse al principio de esta función, por lo que cualquier programa debe existir una función “main” en alguna parte de dicho programa.

Esta función principal llamará a otras funciones que la ayuden a realizar su trabajo, algunas de estas funciones son de la biblioteca de C, y otras las puede realizar usted mismo.

Una de las formas en que pueden comunicar datos entre las diversas funciones es por medio de una lista de valores que proporciona la función que llama, a la función que está invocando, esta lista de valores o argumentos se ubican entre paréntesis después del nombre de la función.

En nuestro primer programa la función main() está definida para ser una función que no espera argumentos, por lo que la lista entre paréntesis está vacía.

La primera línea del programa: #include <stdio.h> le indica al compilador C que incluya la biblioteca estándar de entrada/salida, con esta línea empiezan muchos de los programas escritos en lenguaje “C”.

Para una mejor comprensión de nuestro ejemplo explicaremos el significado de cada una de las líneas escritas:

#include <stdio.h>	Incluye información de la biblioteca estándar de E/S
main()	Definición de la función main que no recibe argumentos
{	Todas las instrucciones de una función se encierran entre llaves “{ }”
printf(“Buenos días amigos\n”);	Desde la función main se llama a la función de biblioteca printf, la cual imprime la secuencia de caracteres encerrada entre comillas; \n representa el carácter “línea nueva”
}	

ACTIVIDAD DE APRENDIZAJE

El estudiante deberá escribir el programa, compilarlo, ligarlo para obtener el objetivo deseado en los equipos de cómputo de la escuela, entregará al profesor el programa impreso y los resultados obtenidos.

En los capítulos siguientes iremos aprendiendo operaciones, estructuras, funciones, argumentos, variables, etc., pero todos los programas que realicemos deben cumplir con la estructura básica descrita en este capítulo.

AUTOEVALUACIÓN

1. Menciona las principales características del Lenguaje de Programación “C”

    - *Lenguaje de propósito general.*
    - *Tecnología abierta, es decir no depende del equipo o del sistema operativo.*
    - *Alta velocidad de ejecución.*

2. ¿En qué consiste el proceso de “Compilar” un programa desarrollado en un lenguaje de alto nivel?

    Sirve para convertir el programa escrito en lenguaje de alto nivel a código de máquina (unos y ceros) entendible para la computadora, además de verificar la sintaxis del programa.

3. ¿Qué instrucción del lenguaje de programación “C” sirve para llamar la biblioteca estándar de entrada/salida?

    #include <stdio.h>

4. ¿Cuál es el nombre de la función principal, con la cuál empieza y termina un programa desarrollado con “C”?

    Main

5. ¿Entre qué símbolos deben escribirse todas las instrucciones a jecutarse en una función desarrollada en “C”?

    Todas las instrucciones de una función deben encerrarse entre llaves { }

UNIDAD 2

TIPOS DE DATOS SIMPLES

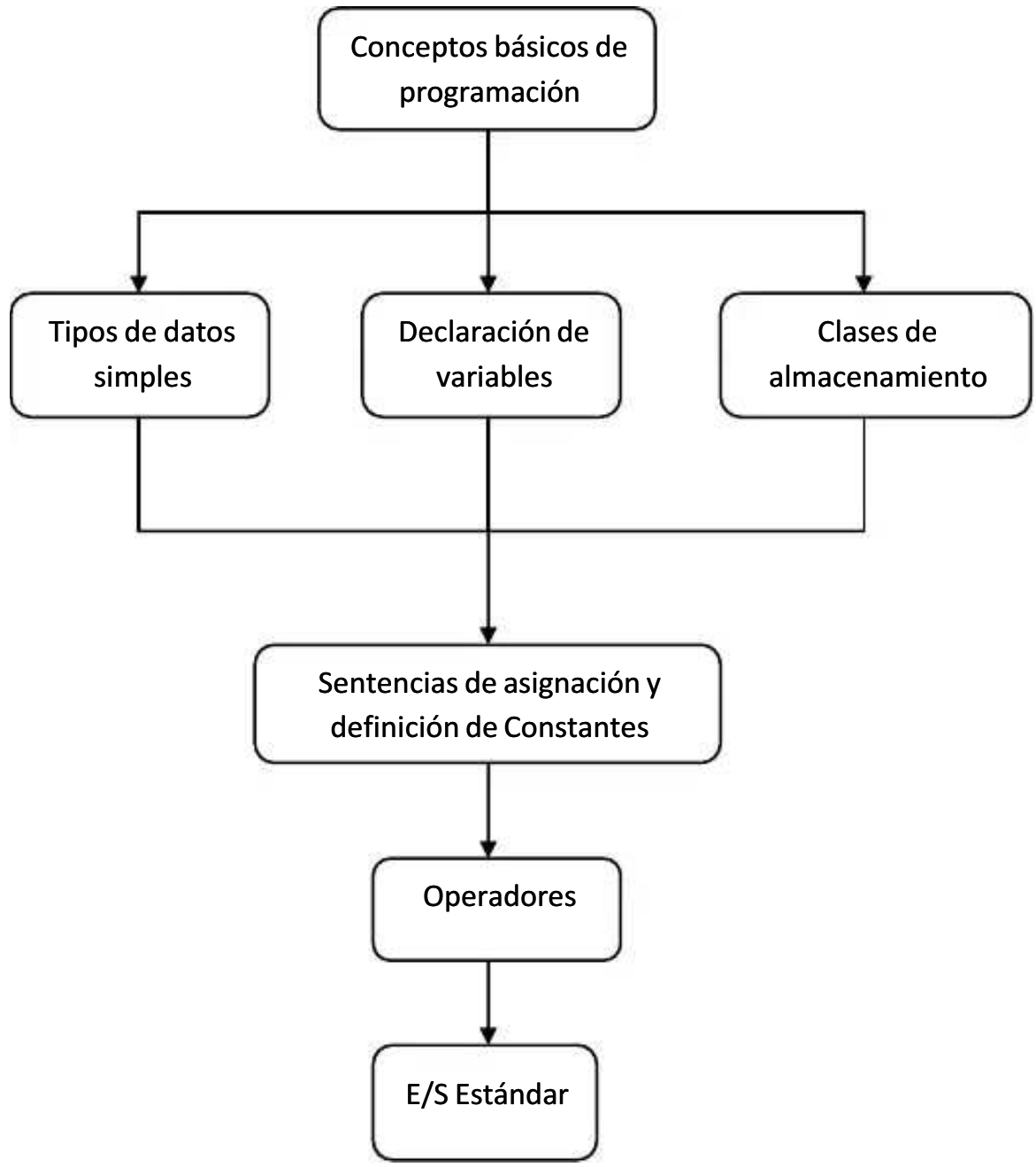
OBJETIVO

Que el estudiante aprenda los diferentes tipos de datos, como se declaran variables, conozca el concepto de constantes y se familiarice con los diferentes operadores y con su manejo, por último aprenderá los diferentes procedimientos existentes en el lenguaje de programación “C”.

Sugerimos recursos, actividades de aprendizaje y formas auto-evaluativas de progresiva dificultad, estos elementos son flexibles y adaptables; por tanto, no todos deben efectuarse, solo aquellos que el profesor estime necesarios para el logro de los objetivos del curso. De esta manera adecuará este capítulo, y el libro en general a sus necesidades.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

El lenguaje “C” se desarrolló para el manejo sencillo de datos, con variables, constantes y operadores simples, si bien permite el manejo de información numérica y alfanumérica, las operaciones son sencillas, dentro de este capítulo

aprenderemos que las variables y las constantes son los objetos de datos básicos que se utilizan en un programa en “C”, para la utilización de los cuáles se manejan:

- Las declaraciones: muestran las variables que se van a utilizar, establecen su tipo y en muchas ocasiones sus valores iniciales.
- Operadores: indican que vamos a hacer con las variables y constantes.
- Expresiones: combinan los datos (variables y constantes) y producen nuevos valores.

2.1 TIPOS DE DATOS SIMPLES

En “C” existen los siguientes tipos de datos, los cuáles aplican tanto para constantes como para variables:

- **char** Un solo byte, contiene un carácter del conjunto de caracteres empleado.
- **int** representa un número entero.
    - Para este tipo de datos existen los calificadores: short y long, por ejemplo:
        - short int nombre;
        - long int nombre;
- **float** punto flotante de precisión normal
- **doublé** punto flotante de precisión doble.

2.2 DECLARACIONES DE VARIABLES

Los nombres de variables se componen de letras y dígitos, también podemos utilizar el guión medio “-”, pero siempre el primer carácter del nombre de una variable debe ser una letra.

Las letras mayúsculas y minúsculas son distintas, por lo que debemos tener cuidado en el uso de letras mayúsculas para evitarnos errores en la programación, por ejemplo **x** y **X** son dos variables distintas.

Por convención en “C” todas las variables usan letras minúsculas y las constantes simbólicas usan letras mayúsculas. Los primeros 31 caracteres de un nombre interno son significativos.

Existen palabras reservadas del lenguaje “C” como son if, else, char, int, etc. que no pueden utilizarse como nombres de variables.

Es muy conveniente que se utilicen nombres que tengan que ver con el propósito de la variable.

Todas las variables deben de ser declaradas antes de ser utilizadas, es decir para que podamos utilizar una variable es preciso utilizar una “declaración” previamente, la declaración especifica un tipo y contiene una lista de una o más variables del tipo indicado, las declaraciones deben tener la siguiente sintaxis:

<tipo> <nombre1, nombre2,…>;

Donde <tipo> toma cualquiera de los valores expresados en el tema 2.1, como ejemplos podemos citar las siguientes declaraciones:

```
int num, alfa, variable;

char nombre, apellido, n;

float x;
```

Como se observa en los ejemplos anteriores todas las declaraciones finalizan con el símbolo “;”

2.3 Clases de almacenamiento

Como mencionamos anteriormente en “C” existen los siguientes tipos de variables:

char Un solo byte, contiene un carácter del conjunto de caracteres empleado 8 bits.

int representa un número entero.

Para este tipo de datos existen los calificadores: short y long, por ejemplo:

short int nombre; entero de 16 bits como máximo

long int nombre; entero de 32 bits como máximo

por lo que un int es de 16 o de 32 bits, pero cada compilador puede seleccionar los tamaños de acuerdo a su hardware

float punto flotante de precisión normal.

double punto flotante de precisión doble.

Existen también los calificadores signed y unsigned, que se utilizan tanto para datos tipo char como para datos int, los números unsigned son siempre positivos o cero.

ACTIVIDAD DE APRENDIZAJE

El estudiante deberá investigar, para el equipo de cómputo y el compilador utilizados en clase, los tamaños disponibles para todos los tipos de datos señalados, incluyendo el uso de los calificadores short, long, unsigned y signed.

2.4 SENTENCIAS DE ASIGNACIÓN

En el lenguaje de programación “C”, la asignación de valores se da mediante el símbolo “=”, como podemos ver en las expresiones siguientes:

i = j +3

z = n*5

El resultado o valor de la expresión del lado derecho del síbolo “=” se le asignará a la variable ubicada en su lado izquierdo.

Expresiones donde la variable del lado izquierdo se repite inmediatamente en el lado derecho, como en la expresión:

i = i + 2

Puede ser escrita en la forma compacta

i += 2

El operador += se llama operador de asignación.

La mayoría de los operadores binarios, es decir que tienen un operando derecho y uno izquierdo, tienen un correspondiente operador de asignación op =, donde op es uno de los siguientes símbolos:

+ - * / % < < > > & ^ □

2.5 Definición de Constantes

Una constante entera es un int, una constante long se escribe con una l o L al final de la constante, un entero demasiado grande para caber dentro de un valor int se toma como long, si la constante no tiene signo se escribe con una u o U y con una ul o UL se escribe una constante long sin signo.

El valor de un entero puede especificarse en forma octal o hexadecimal en lugar de la forma decimal. Un 0 (cero) al principio de una constante entera significa octal; un 0x o 0X al principio significa hexadecimal.

Ejemplo

Decimal	31
Octal	031
Hexadecimal	0X1F

Estas pueden ser seguidas por L para convertirlas en long y U para convertirlas en unsigned (sin signo).

Por ejemplo: 0XFUL es una constante sin signo long con valor de F hexadecimal o 15 en decimal.

Las constante de punto flotante contienen punto decimal (123.45) o un exponente (le-2), o ambos, su tipo es double, los sufijos f o F indican una constante float; l o L indican un long double.

Una constante de tipo carácter se escribe dentro de apóstrofos ‘ ’.

El valor de una constante de carácter es el valor numérico del carácter dentro el conjunto de caracteres de la máquina

Por ejemplo ‘x’, ‘0’ este último ejemplo el valor de ‘0’ en el conjunto de caracteres ASCII es 48 no equivale al entero 0

2.6 Operadores

Los operadores nos permiten el manejo de operandos para obtener nuevos valores, generalmente son binarios, es decir requieren 2 operandos, pero también existen en “C” operadores que solo utilizan un operando.

Como ejemplo de operadores podemos utilizar los operadores de asignación como +=

```
i += 2
```

En esta expresión se incrementa en 2 el valor de i y es equivalente a la expresión:

```
i = i + 2
```

La mayoría de los operadores binarios tienen un correspondiente operador de asignación, *op=* donde op es uno de los siguientes operadores binarios:

```
+  -  *  /  %  <<  >>  &  ^  |
```

Veamos el siguiente ejemplo: si exp1 y exp2 son dos expresiones

exp1 op= exp2

Es equivalente a:

expr1 = (expr1) op (expr2)

La expresión: x *= y+1

Significa: x= x*(y+1)

2.6.1 Operadores aritméticos

Los operadores aritméticos binarios, es decir que llevan 2 operandos, son:

+, -, *, / y el operador módulo %

Este último produce el residuo de una división, por ejemplo:

x % y produce el residuo cuando x se divide entre y.

Por lo que da el valor cero cuando y divide a x exactamente, no se aplica a operandos float o double.

2.6.2 Operadores de relación y lógicos

Los operadores de relación son:

> >= < <=

Todos tienen la misma prioridad, bajo estos operandos en prioridad están los operadores de igualdad:

== ¡=

Estos operadores (de relación) tienen prioridad inferior a los operadores aritméticos, es decir primeramente se realiza la operación aritmética y luego la operación de relación, por ejemplo, la operación:

i < a – 1

es equivalente a la siguiente operación:

i < (a – 1).

Los operadores lógicos son:

&& y ||

Todas las expresiones que usan estos operadores se evalúan de izquierda a derecha, deteniéndose la evaluación tan pronto como se obtenga el resultado "verdadero" o "falso"

ACTIVIDAD DE APRENDIZAJE

El estudiante deberá realizar un programa sencillo donde utilice los diferentes operadores y validar la preferencia que ellos tienen en el cálculo de los resultados, el maestro puede dejar un programa en particular a realizar o puede dejar la opción abierta para que los estudiantes decidan qué problema resolver.

2.6.3 Conversiones de tipo

Cuando un operador tiene diferentes tipos de operandos, estos se convierten a un tipo común de acuerdo a las siguientes reglas:

Las únicas conversiones automáticas son las que se convierten un operando "angosto" en uno "amplio", sin pérdida de información, por ejemplo convertir un

entero en punto flotante (f + i).

Las expresiones que no tienen sentido no son permitidas.

Las expresiones que podrían perder información, por ejemplo de un tipo punto flotante a un entero, pueden producir una advertencia pero son permitidas.

Un char, como es un entero pequeño pueden ser utilizadas libremente en expresiones aritméticas.

A manera de ejemplo podemos analizar la siguiente función, que hemos llamado atoi, convierte una cadena de dígitos en su equivalente numérico.[3]

```
/* atoi: convierte s en entero */
int atoi (char s[])
{
        int i, n;
        n=0;
        for (I = 0; s[i] >= ‘0’ && s[i] <= ‘9’; ++i)
            n = 10*n+(s[i]-‘0’);
        return n;
}
```

ACTIVIDAD DE APRENDIZAJE

El estudiante deberá referirse al libro “El lenguaje de programación C” autores Brian W. Kernighan y Dennis M. Ritchie editorial Pearson Educación , página 47 para analizar cada una de las expresiones utilizadas en este ejercicio y las entregará al profesor.

---

[3] Ejemplo tomado del libro: El Lenguaje de Programación C de Brian W. Kernighan, Dennis M. Ritchie

En resumen, y en forma general las siguientes reglas bastarán:

- Si cualquiera de los operandos es long doublé, el otro se convierte en long doublé.
- Si cualquiera de los operandos es doublé, el otro se convierte en doublé.
- Si cualquier operando es float, el otro se convierte a float.
- Los tipos chart y short se convierten a int.
- Si cualquier operando es long el otro se convierte a long.

2.6.4 Operadores de incremento y decremento

En el lenguaje de programación “C” se introducen dos operadores poco comunes que incrementan y decrementan variables:

++ suma 1 a su operando.

-- resta 1 a su operando

Ejemplo: n++

Se pueden utilizar antes o después del operando, pero si se utiliza antes (++n) incrementa el operando antes que su valor se utilice, y se se emplea después (n++) incrementa n después de incrementar su valor.

Ejemplo:

Si n=5

X=n++ asigna 5 a la variable x

X=++n asigna 6 a la variable x

2.6.5 Operadores para manejo de bits

Existen los siguientes 6 operadores para el manejo de bits:

&	AND de bits
|	OR inclusivo de bits.
^	OR exclusivo de bits.
<<	Corrimiento a la izquierda.
>>	Corrimiento a la derecha.
˜	Complemento a uno.

Para que la prioridad de los diferentes operadores utilizados en “C” nos quede más clara, analice la siguiente tabla:

Máxima	() [] -- →
	¡ ˜ ++ -- -(type) *& size of
	* / %
	+ -
	<< >>
	< <= > >=
	== ¡=
	&
	^
	|
	&&
	||

	A?:
Mínima	= += -= *= /=

2.7 PROCEDIMIENTOS DEFINIDOS DE ENTRADA / SALIDA ESTÁNDAR

Recuerde que en el lenguaje “C” no existen funciones construidas e incorporadas para realizar operaciones de E/S, pero estas funciones si se encuentran en la librería estándar de “C”.

En este capítulo analizaremos las siguientes funciones:

getchar()	lee un carácter desde la entrada estándar (normalmente teclado.
putchar()	imprime un carácter por la salida estándar (normalmente la pantalla.
gets()	Lee una cadena desde el teclado
puts()	Escribe una cadena en la pantalla
printf()	produce salida formateada.
scanf()	produce entrada formateada.

Con estas dos últimas funciones se puede formatear la información.

En “C” toda E/S es orientada a carácter, esto aplica para la lectura y escritura por consola (teclado y pantalla), y para las funciones de archivos en disco, es decir en “C” uno puede leer y escribir bytes.

La entrada y salida por consola se refiere a las operaciones sobre el teclado y la pantalla de la computadora.

El mecanismo de entrada más simple es el de leer un carácter a la vez de la entrada estándar, normalmente el teclado con la función getchar:

int getchar(void)

getchar regresa el siguiente carácter de la entrada cada vez que es invocada o EOF cuando encuentra el fin de archivo. La constante simbólica EOF (End of File) está definida en <stdio.h> y su valor típicamente es -1.

Un archivo puede tomar el lugar del teclado empleando la convención < para re direccionamiento de entrada

La función putchar escribirá un carácter en la salida estándar, normalmente la pantalla de la computadora.

Las funciones gets() y puts(), son utilizadas para leer e imprimir cadenas de caracteres por consola, gets() devuelve una cadena de caracteres terminada con el carácter nulo, permite corregir errores en la cadena mediante la tecla “backspace” antes de oprimir la tecla “enter”.

puts() escribe el argumento de cadena sobre la pantalla reconoce los códigos de barra invertida, al igual que printf, como \n para cambio de línea.

La función printf permite salidas formateadas y tiene la siguiente sintaxis:

Printf(“cadena de control” , lista de argumentos);

Donde la cadena de control consta de dos tipos de elementos, el primero está formado por los caracteres que se imprimirán en la salida estándar y el segundo contiene comandos de formato, que muestra la forma en se que se mostrarán en la salida estándar los argumentos siguientes, debe haber el mismo número de comandos de formato que de argumentos

Los códigos de control de formato de printf son:

Código de printf()	formato
%c	un único carácter.
%d	Decimal.
%e	Notación científica.
%f	Coma flotante decimal.

%g	Utiliza %e o %f, la que sea mas corta.
%o	Octal.
%s	Cadena de caracteres.
%u	Decimal sin signo.
%x	Hexadecimal.

Estos códigos de control de de formato pueden incluir modificadores para especificar: anchura de campo, número de decinmales y un indicador que ajuste a la izquierda. Un entero situado entre el signo % y el comando de formato actúa como indicador de anchura mínima del campo

Ejemplos de salidas formateadas con printf()

Sentencia printf()	Salida
(“%-5.2f”,123.234)	123.23

ACTIVIDAD DE APRENDIZAJE

El estudiante deberá realizar el siguiente programa en “C”, el cuál tomará caracteres desde el teclado y los imprimirá en pantalla, cambiando las mayúsculas por minúsculas y viceversa, para finalizar el programa se debe pulsar un punto.

```
Main () /* cambia mayúsculas por minúsculas y viceversa*/
{
        char ch;
        do {
                ch=getchar();
                if (islower(ch)) putchar(toupper(ch));
                else putchar(tolower(ch));
        }while (ch !='.'; /* use el punto para parar el programa*/
}
```

El alumno buscará la definición de las funciones islower, toupper y tolower para discutirlas en clase.

Nota las funciones do-while, if-else se estudiarán a lo largo de este curso.

AUTOEVALUACIÓN

1. Menciona los operadores que maneja C

    - *Operadores aritméticos.*
    - *Operadores lógicos y de relación.*
    - *Operadores de manejo de bits.*
    - *Operadores de asignación.*

2. Menciona los diferentes tipos de variables que se manejan en C.

    char

    int.

    short int

    long int

    float

    double.

3. ¿Cuál es el símbolo que se utiliza para asignar un valor a una variable?

    =

4. Menciona un ejemplo de operador de asignación

    +=

5. Menciona un operador de incremento y uno de decremento

    ++ --

UNIDAD 3

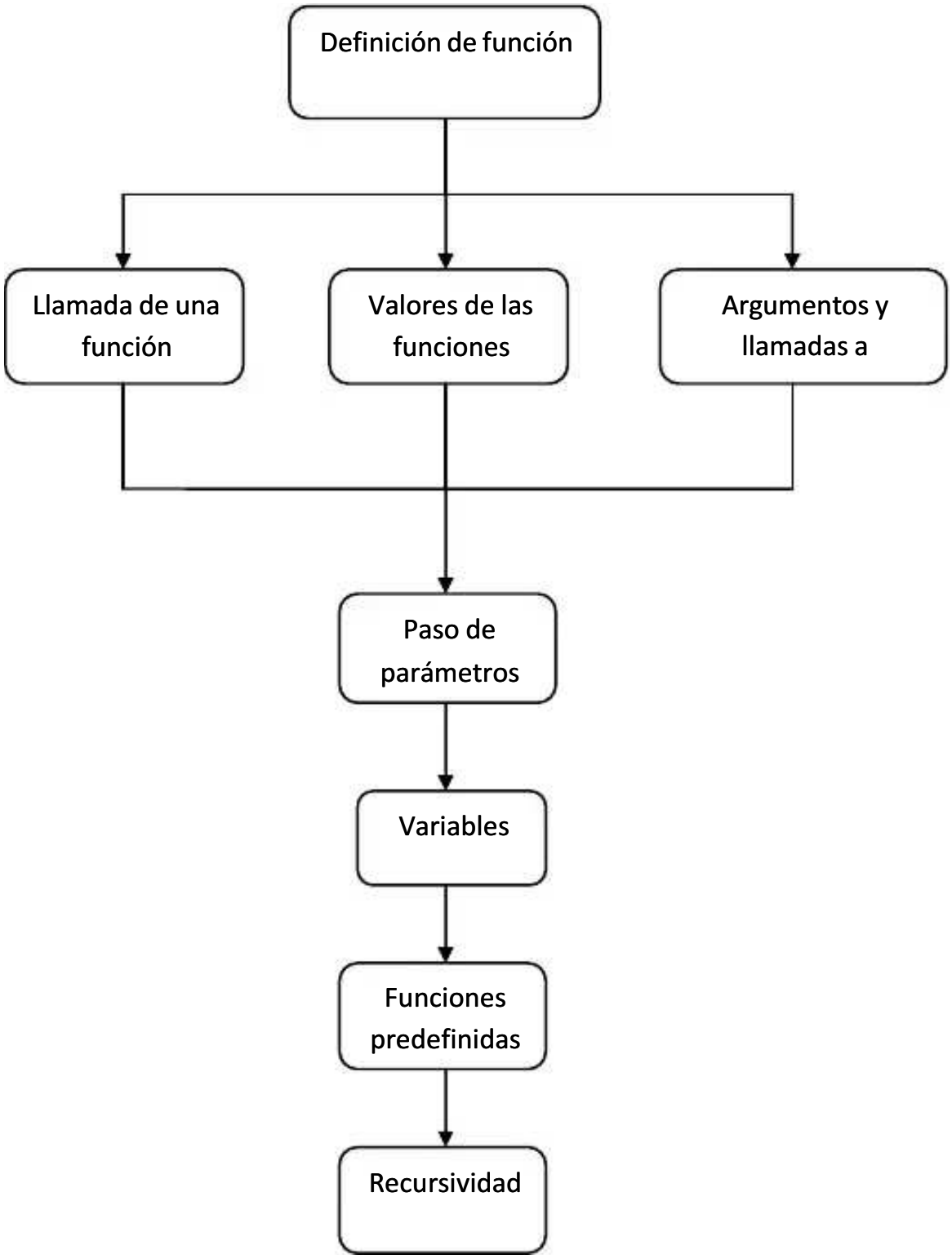
FUNCIONES Y LA ESTRUCTURA DEL PROGRAMA

OBJETIVO:

Que el estudiante aprenda qué son las funciones y cómo utilizarlas, cómo pasar y recibir información a y desde una función, conozca las diferentes variables que existen en “C” y conozca el concepto de recursividad.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

Las funciones nos sirven para dividir tareas grandes en tareas más pequeñas, nos dan la posibilidad de construir sobre los programas que otros han hecho y utilizarlos para nuevas soluciones en lugar desde comenzar desde cero.

El lenguaje “C” se diseñó para que las funciones fueran eficientes y muy fáciles de utilizar normalmente los programas que se escriben en “C” se componen de muchas funciones pequeñas en lugar de solo algunas grandes.

3.1 DEFINICIÓN DE FUNCIÓN

Las funciones son bloques con los que se constituyen programas en lenguaje “C” y en ellas se llevan a cabo las funciones del programa.

Una vez que una función ha sido escrita, compilada, ligada y depurada puede utilizarse una y otra vez desde cualquier otro programa

3.2 LLAMADA DE UNA FUNCIÓN

El formato general de una función en “C” es el siguiente:

```
nombre_funcion (lista de parámetros)
declaraciones de parámetros;
{
        cuerpo de la función;
}
```

El número de parámetros pueden ser cero, en este caso no es necesaria las sección de declaraciones.

Todas las funciones devuelven por lo menos un valor, este valor puede estas explícitamente especificado con la palabra return, o bien puede ser cero si no se especifica ningún otro valor. Por default todas las funciones devolverán valores enteros pero se pueden especificar otro tipo de valores.

Una función puede ser utilizada dentro del cuerpo de una expresión, debido a que cada función devuelve un valor, como ejemplo de esto mencionamos las siguientes expresiones todas validad en “C” :

```
x= potencia (y);

if ( max(x,y) > 100) printf(“mayor que”);
```

3.3 FUNCIONES QUE REGRESAN VALORES NO ENTEROS

El valor entero es el tipo de dato que por default se devuelven en funciones de “C”, en ocasiones es necesario que devuelvan otro tipo de datos, por esto las funciones se pueden declarar para que devuelvan cualquier tipo de datos incluidos en “C”. La forma de la declaración es similar al de declaración de variables, se utilizan el mismo tipo de especificadores, precediento al nombre de la función.

El formato general de una declaración de función es:

```
Especificador_tipo  nombre_función(lista de parámetros)
        Declaraciones de parámetros;
        {
                Expresiones  (sentencias de la función);
        }
```

El especificador de tipo le indica al compilador “C” el tipo de dato que va a devolver la función.

Cuando se utiliza un valor devuelto que no sea entero se tiene que hacer una segunda cosa: La rutina que llamó a la función tiene que saber el tipo de dato que devolverá la función, por lo que debemos declarar la función dentro de la rutina de llamada.

3.4 Argumentos, llamada de una función por valor

Generalmente existe dos formas en que pueden pasarse los argumentos a las funciones: La primera se denomina “llamada por valor”, este método copia el valar de cada uno de los argumentos en los parámetros formales de la función, la segunda forma se conoce como “llamada por referencia” con este método, la dirección de cada argumento se copia los parámetros de la función, es decir los cambios hechos en el parámetro afectaran a la variable utilizada para llamar a la función.

Las funciones “C” utilizan la llamada por valor, esto significa que no se pueden alterar las variables utilizadas para llamar a la función esta es la forma general.

3.5 Paso de parámetros de una función

Los parámetros dentro de una función “C” se pasan después del nombre de la función y deben ser encerrados entre paréntesis () y a continuación del nombre se deben declarar los parámetros de la función.

3.6 Variables

Una variable local es dinámica; se crea cuando la función se ejecuta y se destruye en el momento en que la función termina, una variable local solo se conoce dentro de la función en la que es declarada.

3.6.1 Variables externas

Una variable externa se declara fuera de cualquier función y es conocida por todas las funciones del programa., es decir están disponibles para todas las funciones Las variables externas permanecen durante toda la duración del programa, cualquier función puede tener acceso a variables externas haciendo referencias a ellas solamente por su nombre.

3.6.2 Variables Estáticas

Una variable estática mantendrá su valor entre una llamada y otra a la función, solo se conoce por su función y permanecerá mientras lo haga el programa.

3.6.3 Variables registro

Una declaración register indica al compilador “C” que la variable en cuestión se empleará constantemente, este tipo de variables deben ser colocadas en registros de la máquina, lográndose programas mas pequeños y rápidos.

La declaración register se realiza de la siguiente manera:

```
register int x;

register char c;
```

Solo algunas variables de una función se pueden mantener en registros, el compilador puede ignorar la sugerencia de register.

3.7 FUNCIONES PREDEFINIDAS EN “C”

En “C” existen librerías con una gran cantidad de funciones predefinidas, las funciones, tipos y macros de la biblioteca estándar están declarados en encabezados o headers estándar:

<assert.h>	<float.h>	<math.h>	<stdarg.h>	<stdlib.h>
<ctype.h>	<limits.h>	<setjmp.h>	<stddef.h>	<string.h>
<errno.h>	<locale.h>	<signal.h>	<stdio.h>	<time.h>

Referirse apéndice B de libro “El lenguaje de programación C”[4] para una definición completa de las principales funciones de las librerías estándar de “C”

ACTIVIDAD DE APRENDIZAJE

El estudiante analizará por lo menos 10 funciones predefinidas de “C”, las cuales forman parte de la librería estándar del lenguaje.

Estas deberán ser discutidas en clase, tratando de mencionar diferentes tipos de funciones, es decir de entrada y salida, de operaciones para archivos, funciones de error, funciones para cadenas de caracteres, funciones matemáticas, etc.

3.8 RECURSIVIDAD

En los programas desarrollados en “C” las funciones pueden llamarse a si mismas es decir son recursivas, esto es si dentro del cuerpo de la función existe

[4] El Lenguaje de programación C, autores Brian W Kernighan y Dennis M. Ritchie, editorial Pearson Educación

una expresión donde se llama a la misma función, se podría conocer también como definición circular.

Un ejemplo de una función recursiva se muestra a continuación

```
factr(n)        /* recursiva */

int n;

{

        int respuesta;

        if(n==1) return(1);

        respuesta=factr(n-1)*n;

        return(respuesta);

}
```

AUTOEVALUACIÓN

1. Explica que son las variables externas en una función escrita en “C”

   Una variable externa se declara fuera de cualquier función y es conocida por todas las funciones del programa. Las variables externas permanecen durante toda la duración del programa.

2. ¿En qué consiste el concepto de recursividad?

   Es la capacidad que tienen las funciones desarrolladas en “C” de llamarse a si mismas.

3. Menciona 3 funciones de la biblioteca estándar de “C”

   printf *getchar* *putchar*

4. ¿En qué consiste la llamada a una función por valor?

   copia el valar de cada uno de los argumentos en los parámetros formales de la función.

5. Indique los tipos de datos que pueden ir precedidos del modificador register

   char e int

UNIDAD 4

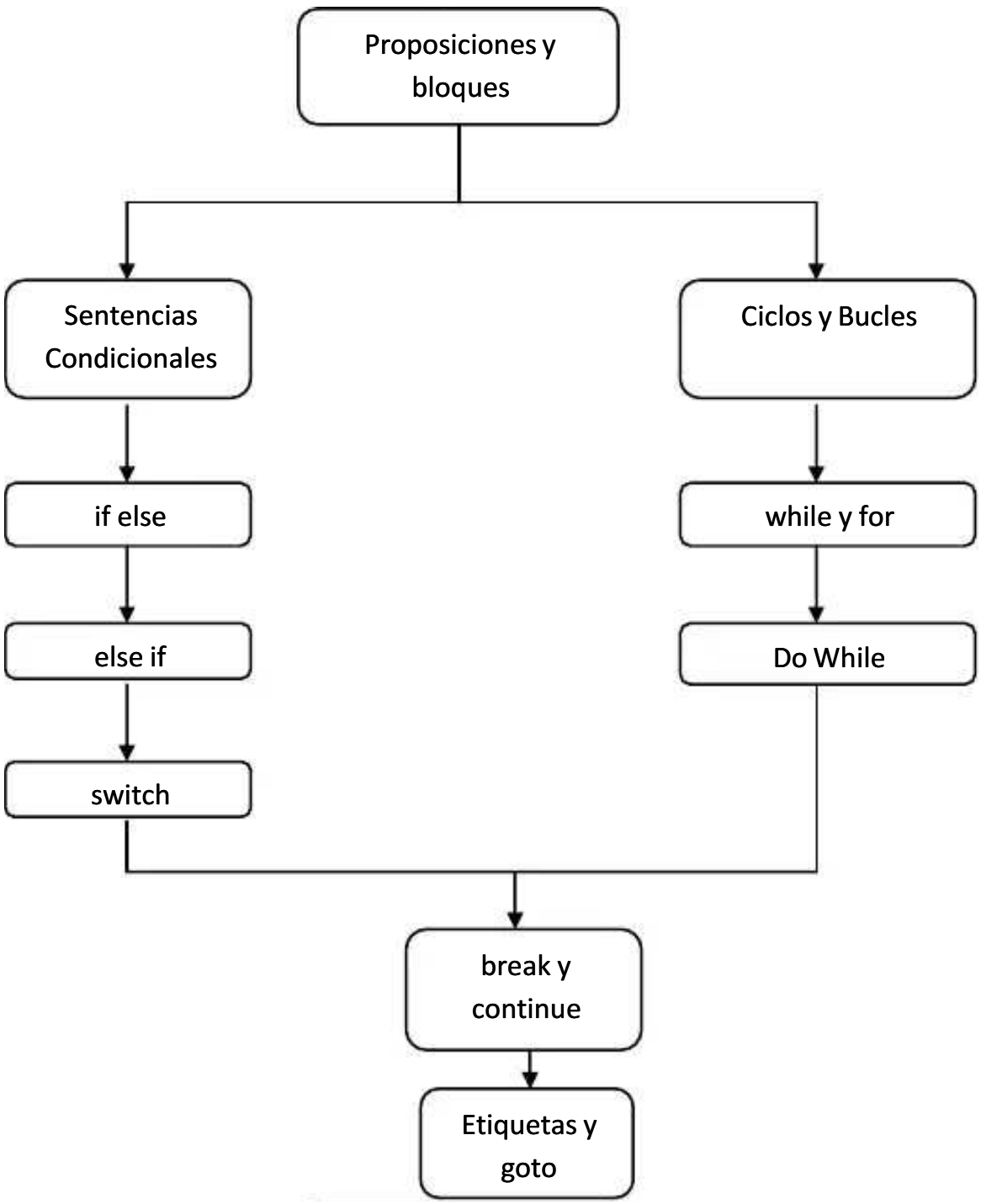
ESTRUCTURAS DE CONTROL DE FLUJO

OBJETIVO:

Que el estudiante aprenda a utilizar las sentencias condicionales, los ciclos, las proposiciones de rompimiento de la secuencia normal de ejecución de un programa (break, continue, goto y etiquetas) para la toma de decisiones en la elaboración de programas con el lenguaje de programación “C”, al finalizar este capítulo podrá realizar programas más complejos para la solución de los problemas que se le planteen.

TEMARIO

Mapa Conceptual Unidad 4



INTRODUCCIÓN

El lenguaje “C” cuenta con una serie de expresiones para poder controlar la secuencia en la ejecución de los diversos bloques que componen el programa, se cuenta con sentencias condicionales y con ciclos o bucles

- Las sentencias condicionales permiten la ejecución de solo un bloque de instrucciones basándonos en el resultado de una condición.

- Los ciclos o bucles nos permiten la repetición de un bloque de instrucciones hasta que una condición se cumpla.

4.1 PROPOSICIONES Y BLOQUES

En “C” una expresión como x=0 o i++ o cualquier otra, se convierte en una proposición cuando va seguida de un punto y coma “;”, por ejemplo en las siguientes expresiones:

x=0;

¡++;

En el lenguaje “C” el punto y coma es el terminador de proposición, no cumple funciones de separador como en otros lenguajes.

Las llaves { } se ocupan para agrupar declaraciones y proposiciones dentro de un bloque, es decir dentro de una proposición compuesta, por lo que son equivalentes, sintácticamente hablando, a una proposición sencilla. Como ejemplo podemos citar:

- Las llaves que encierran todas las proposiciones de una función.
- Las llaves que encierran proposiciones múltiples com en el caso de if, else, while, for.

Algunas características importantes de los bloques son:

- Dentro de un bloque se pueden declarar variables.
- No se debe escribir un punto y coma después de la llave derecha “}” que cierra el bloque.

Las declaraciones de variables pueden seguir a la llave izquierda “{“ que inicia cualquier proposición compuesta, siendo éstas internas al bloque donde son declaradas, y no tienen relación alguna con variables declaradas en bloques externos, y permanecerán hasta que se encuentre la llave derecha “}” que cierra el bloque, que corresponda con la llave inicial, ejemplo:

```
if ( x> 0) {
```

```
    int i; /* se declara una nueva variable entera i */
    for (i=0; i<n; i++)
    …….
    ……
}
```

Recordemos que un comentario se puede escribir al finalizar una proposición, como sucede en la declaración de la variable entera i, siempre y cuando lo encerremos entre “/*” y “*/” todo lo que va entre estos símbolos es considerado un comentario y no será analizado por el compilador “C”.

La variable i solo estará disponible en la rama “verdadera” del if, por lo que esta i no tiene ninguna relación con alguna otra variables llamada i fuera de este bloque, su valor se pierde en el momento de encontrar la llave derecha “}” que indica terminación del bloque.

Una variable automática que se declara e inicializa en un bloque, deberá ser inicializada cada vez que se entra al bloque

Una variable estática se inicializa solamente la primera vez que se entra al bloque.

Como un ejemplo final analicemos las siguientes proposiciones:

```
Int a
Int b
f1 (double a)
{
    double b;
```

```
    ..............
}
```

En este ejemplo

En la función f1 las ocurrencias de a se refieren al parámetro que es double.

Fuera de f1 se refieren al int externo, lo mismo ocurre para la variable b.

Se recomienda evitar nombres de variables dentro de una función que sean idénticos con nombres de un alcance exterior, para evitar errores o confusiones.

ACTIVIDAD DE APRENDIZAJE

El estudiante deberá realizar un programa que utilice una variable i externa y una variable i interna a un bloque de instrucciones, realizar variaciones del valor de dicha variable, tanto fuera como dentro del bloque, mandando a imprimir su valor varias veces para verificar que son dos variables independientes.

4.2 Sentencias Condicionales

Las sentencias condicionales nos sirven para poder elegir la ejecución de uno u otro bloque en base al resultado de una condición previamente establecida, es decir podemos escoger uno de dos caminos en base a que se cumpla o no una condición.

4.2.1 If Else

Esta proposición se utiliza para tomar decisiones, su sintaxis es:

```
if (expresión)
        proposición1
else
        proposición2
```

La expresión entre paréntesis ( ) se evalúa, si es verdadera (tiene un valor diferente de 0) se ejecuta la proposición1, pero si es falsa (tiene un valor igual a 0) entonces se ejecuta la proposición2, esto siempre que exista una parte de else, ya que esta última parte es opcional.

Como un if prueba el valor numérico de una expresión, podemos abreviar el código, por ejemplo:

```
Utilizar:       if (expresión)
En lugar de:    if (expresión ¡= 0)
```

Como la parte del else es opcional, en ocaciones cuando existen if's anidados puede existir cierta ambigüedad al escribir un else, esto se resuelve al asociar el else con el if anterior, es decir con el mas cercano,es recomendable utilizar llaves { } para forzar la asociación correcta, por ejemplo:

```
if (a>0)
        if (n>m)
                x=n;
        else
                x=m;
```

El else corresponde al if más interno es decir si no se cumple la expresión (n>m) entonces se ejecutarán las expresiones del else. Si lo que se

quiere es que el else se ejecute si la expresión (a>0) no se cumple (Asociado al primer if) se usarán llaves:

```
if (a>0) {

        if (n>m)
                x=n;

        }

else

        x=m;
```

Por claridad en la programación podríamos usar un comentario al final de cada if else para señalar su final:

```
        /* End if */
```

Por ejemplo:

```
if (a>0)

        if (n>m)

                x=n;

        else

                x=m;

        /*end if*/

/*end if*/
```

De esta manera evitamos ambigüedades, o falta de claridad en la programación.

4.2.2 Else If

Para poder contar con una desición múltiple, es decir poder evaluar una condición y en base al resultado de la misma tomar un camino determinado se utiliza esta instrucción, su sintaxis es la siguiente:

```
if (expresión)
        proposición
else if (expresión)
        proposición
else if (expresión)
        proposición
else if (expresión)
        proposición
else
        proposición
```

Todas las expresiones se evaluarán en orden, si cualquiera de ellas es verdadera, la proposición asociada con ella se ejecuta, terminando con ello la cadena de decisiones, como hemos mencionado, cada proposición puede ser una proposición simple o un grupo de ellas dentro de llaves “{….}”.

El último else, en el ejemplo anterior, maneja el caso “ninguno de los anteriores”, es el caso por default que se ejecutará cuando ninguna de las condiciones establecidas se satisface, no siempre se utiliza, pero nos resulta útil para detectar errores o para una condición “imposible”

ACTIVIDAD DE APRENDIZAJE

El profesor deberá solicitar a los alumnos la realización de, por lo menos 2 programas que resuelvan mediante el uso de if-else y/o de else-if problemas sencillos pero que sirvan para comprender completamente el uso de estas sentencias condicionales.

4.2.3 Switch

Esta es una decisión múltiple que prueba si una expresión coincide con unos de un número de valores constantes enteros, su sintaxis es la siguiente:

```
switch (expresión) {

    case exp-const:proposiciones

    case exp-const: proposiciones

    default: proposiciones

}
```

Cada case se etiqueta con uno o más valores constantes enteros o expresiones constantes enteras, cuando un case coincide con el valor de la expresión, la ejecución comienza en ese punto, es importante mencionar que todas las expresiones case deben ser distintas.

La etiqueta default se ejecuta si ninguno de los otros casos se satisfizo, y es optativo, si esta etiqueta no se escribe y ninguno de los casos coincide no se tomará acción alguna.

ACTIVIDAD DE APRENDIZAJE

Los alumnos realizarán, compilarán y ejecutarán el siguiente programa[5].

```
#include <Stdio.h>

main () /* cuenta dígitos, espacios en blanco y otros caracteres */
        {
                int c, i, nwhite, nother, ndigit[10];
                nwhite = nother = 0;
                for(i=0; i<10; i++)
                        ndigit[i] = 0;
                while ((c = getchar()) != EOF) {
                        switch ( c) {
                        case '0': case '1': case '2': case '3': case '4':
                        case '5': case '6': case '7': case '8': case '9':
                                ndigit[c-'0']++;
                                break;
                        case ' ':
                        case '\n':
                        case '\t':
                                nwhite++;
                                break;
                        default:
                                nother++;
                                break;
                        }
                }
                printf("digitos =");
                for (i=0; i<10; i++)
                        printf("%d", ndigit[i]);
```

[5] Tomado del libro El Lenguaje de programación C; autor Brian W. Kernighan, Dennis M Ritchie; editorial Pearson Educación

```
        printf(",espacios blancos = %d, otros= %d\n",
                nwhite, nother);
        return 0;
}
```

4.3 CICLOS Y BUCLES

Los ciclos también llamados bucles se utilizan para repetir proposiciones o bloques mientras cierta condición se cumpla, en la actividad de aprendizaje anterior utilizamos estos ciclos para poder resolver el problema, ahora los explicaremos a detalle.

4.3.1 While y For

La sintaxis de estas expresiones es la siguiente:

```
while (expresión)
        proposición
```

La expresión es evaluada, si es diferente de cero, se ejecuta la proposición y se vuelve a evaluar la expresión, este ciclo se repite hasta el momento en que la expresión es igual a cero, en dicho momento se suspende la ejecución y se continúa el desarrollo del programa a continuación de la proposición.

```
for (expresión1; expresión2; expresión3)
        proposición
```

Normalmente los 3 componentes del ciclo for son expresiones, expresión1 y expresión3 son asignaciones o bien llamadas a funciones y

expresión2 es una expresión de relación, se puede omitir cualquiera de las 3 expresiones pero se deben mantener los “**;**”

El ciclo for escrito anteriormente equivale al siguiente ciclo while:

```
expresión1;
while (expresión2) {
        proposición
        expresión3;
}
```

4.3.2 Do While

Su sintaxis es la siguiente:

```
do
        proposición
while (expresión);
```

La proposición se ejecuta por primera vez, después se evalúa la expresión, si es verdadera entonces se vuelve a ejecutar la proposición, así sucesivamente hasta que la expresión regrese un valor de falso, terminando de esta manera el ciclo.

4.4 Break y Continue

En muchas ocasiones, y dependiendo de cada problema a resolver, es conveniente terminar un ciclo de una manera distinta, es decir que no sea comprobando al inicio o al final del ciclo. La proposición break nos sirve para lograr una salida anticipadapara un ciclo o bucle (for, while o do) o bien un switch.

Un break provoca que el ciclo o switch más interno que lo encierra termine inmediatamente.

4.5 ETIQUETAS Y GOTO

Una excelente funcion de “C” para controlar la secuencia de proposiciones a ejecutar es goto, que en conjunto de las etiquetas dan una importante función a este lenguaje de programación.

Formalmente esta proposición no es necesaria, en la práctica es muy sencillo escribir código de programación sin ella, sin embargo en algunas situaciones puede ser de gran ayuda, la más común consiste en abandonar un procedimiento en una proposición profundamente anidada, es como salir de 2, 3 o más ciclos a la vez. Si utilizamos la proposición Break salimos del ciclo más interno y con goto abandonamos todos los ciclos, reiniciando la ejecución en donde esté indicada la etiqueta.

Por ejemplo:

```
for (…)
        for (…) {
                …
                if (desastre)
                        goto error;
        }
…
error:
        arreglamos el desastre
```

ACTIVIDAD DE APRENDIZAJE

El profesor deberá solicitar a los alumnos la realización de, por lo menos 2 programas que resuelvan mediante el uso de ciclos, break, continue, goto y etiquetas problemas sencillos pero que sirvan para comprender completamente su uso.

AUTOEVALUACIÓN

1. Menciona 2 ejemplos de ciclos o bucles.

    - *while, for, do-while (cualquiera 2 de ellos).*

2. Menciona dos ejemplos de sentencias condicionales.

    - *if-else, else-if, switch (cualquiera dos de ellos).*

3. ¿Qué es un ciclo o bucle?

    - *Es una proposición o bloque que se repite mientras una condición se cumpla.*

4. ¿Para qué nos sirve una sentencia condicional?

    - *Nos permite ejecutar una proposición u otra dependiendo del valor quetoma una condición establecida, pueden ser sentencias condicionales múltiples, es decir con más de 2 tomas de desiciones.*

5. ¿Qué es una proposición en el lenguaje “C”?

    En “C” una expresién cualquier, se convierte en una proposición cuando va seguida de un punto y coma *“;”*

UNIDAD 5

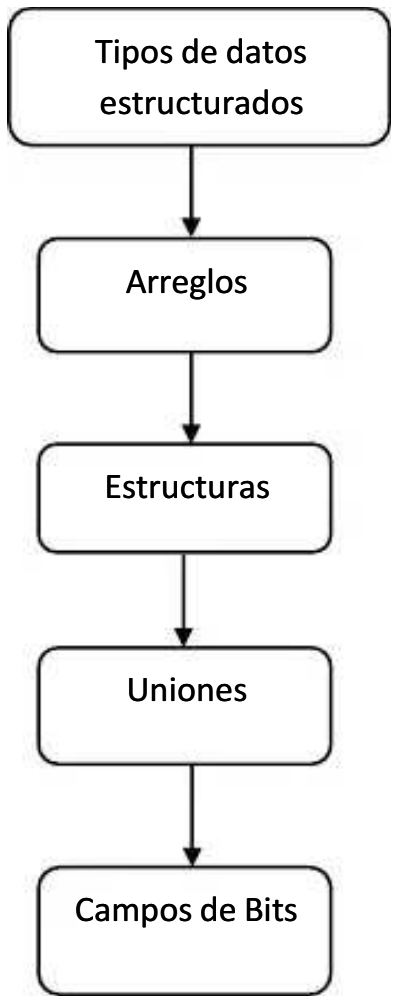
TIPOS DE DATOS ESTRUCTURADOS

OBJETIVO

Que el estudiante aprenda a utilizar arreglos, estructuras, uniones y campos de bits.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

El alumno aprenderá:

- El concepto y manejo de arreglos y estructuras en el lenguaje de programación “C”
- Aprenderá el manejo de uniones y el concepto de campos de bits herramientas importantes en “C”.

5.1 ARREGLOS

Un apuntador es una variable que contiene la dirección de una variable, estos son muy utilizados en “C”, los apuntadores y los arreglos están muy relacionados, en este capítulo analizaremos los arreglos y en el siguiente los apuntadores.

En “C” los arreglos se componen de posiciones contiguas de memoria, donde la dirección más baja corresponde al primer elemento y la más alta al último elemento.

La sintaxis general para un arreglo unidimensional es:

```
type var_nombre[dimensión];
```

Se requiere que a cada arreglo se le dé una longitud en la sentencia de declaración, en “C” los arreglos empiezan en cero, por ejemplo, cuando escribimos:

```
char p[10];
```

Declaramos un arreglo de caracteres que tiene 10 elementos, de p[0} a p[9].

```
char x[10];
```

Declaramos un arreglo de números enteros de 10 elementos de x[0] a x[9].

ACTIVIDAD DE APRENDIZAJE

El estudiante analizará el siguiente ejemplo y lo discutirá en clase:

```
int i[10];

int cont;
for (cont=0;cont<10;++cont) i[cont]=cont;
```

Nota: Recordar la diferencia entre ++var y var++, recordar la estructura del for

---

Los ejemplos anteriores hacen referencia a los arreglos en una dimensión, pero “C” cuenta con arreglos de dos o más dimensiones, es decir con arreglos multidimensionales. Para declarar un arreglo entero bidimensional i con un tamaño de 10,20 debemos escribir:

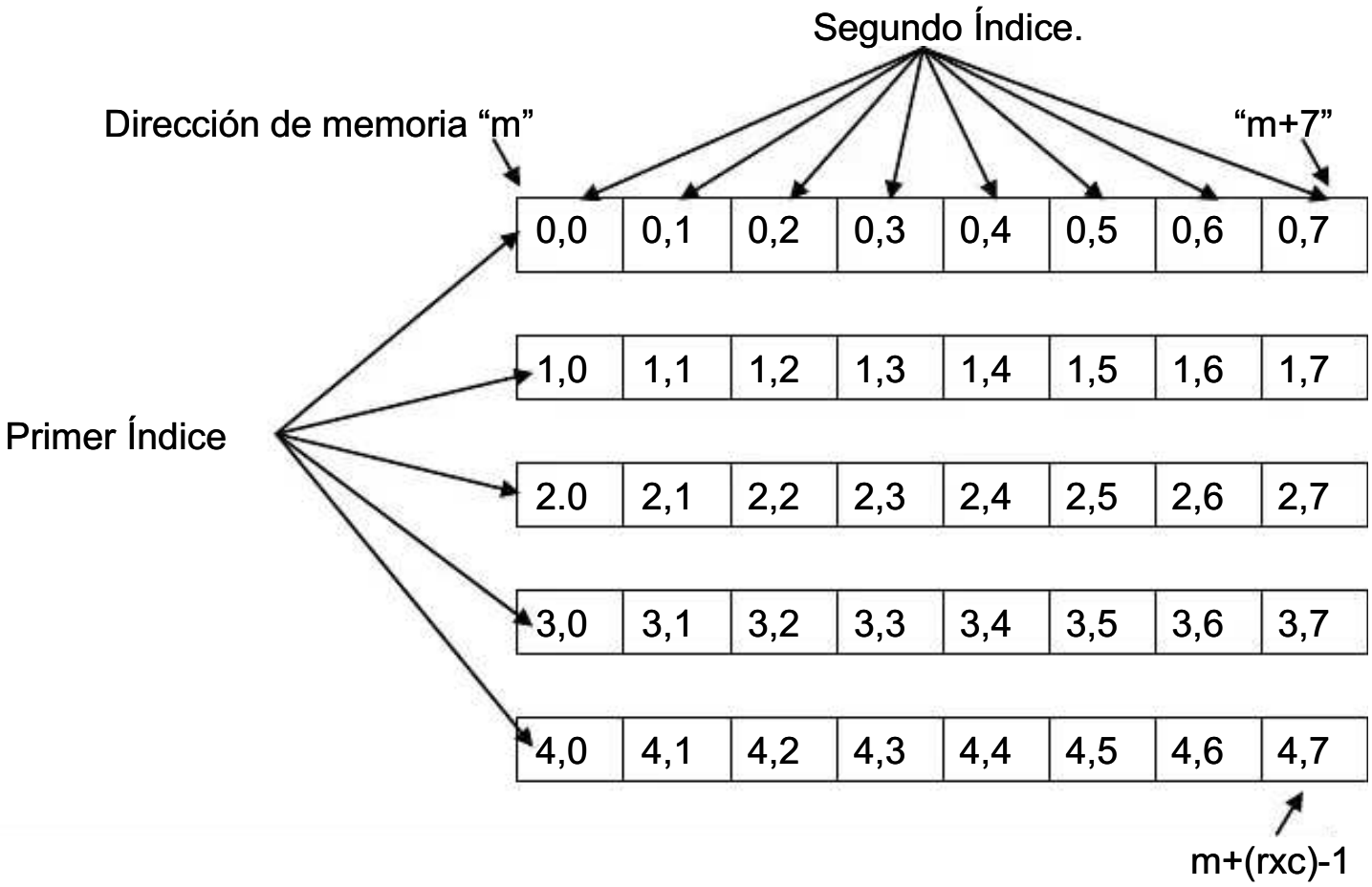
```
int i[10][20];
```

En “C” se indica cada dimensión entre paréntesis cuadrados, no utiliza comas como separadores como se hace en otros lenguajes de programación.

Si queremos hacer referencia a un elemento del arreglo, por ejemplo el 4,6 debemos utilizar la siguiente expresión:

```
d[4][6]
```

Para acabar de entender cómo se guarda un arreglo bidimensional en memoria:



ACTIVIDAD DE APRENDIZAJE

El estudiante realizará un programa para llevar un arreglo multidimensional donde coloque Nombre, Apellido Paterno y Apellido Materno de todos los alumnos inscritos en su grupo.

En clase analizar las diversas soluciones presentadas.

5.2 ESTRUCTURAS

Una estructura en “C” es una colección de una o más variables, de tipos posiblemente diferentes, que se referencian bajo un mismo nombre, con el objeto de mantener en un sitio la información que está relacionada.

En otros lenguajes las estructuras se conocen como “registros”, nos ayudan a organizar datos complicados.

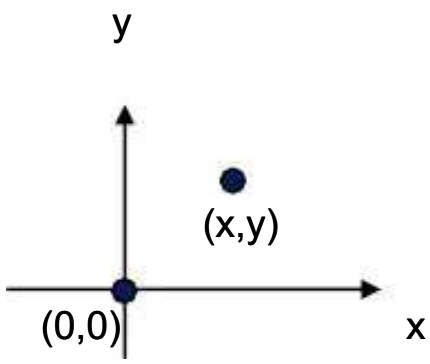
Como ejemplo podemos mencionar el nombre de un apersona y su dirección agrupadas en una lista de correo es un conjunto de información relacionada.

5.2.1 Conceptos básicos de estructuras

Una definición de la estructura forma una plantilla que puede ser utilizada para crear estructuras variables.

Cada estructura se forma de una o más variables que están relacionadas lógicamente, dichas variables se denominan “elementos de la estructura”

Un ejemplo sencillo de estructuras puede ser utilizada para realizar una gráfica cada punto de la misma tiene una coordenada x y una coordenada y, ambas enteras.



Estos dos componentes pueden ser colocados en una estructura declarada como se muestra a continuación:

```
struct point {
        int x;
        int y;
};
```

Con la palabra reservada struct sirve para declarar una estructura, esta es una lista de declaraciones entre llaves, se utiliza un nombre de forma optativa como rótulo de la estructura, en el ejemplo anterior el rótulo de la estructura es “point”, este puede ser utilizado como una abreviatura para la parte de declaraciones entre llaves.

Las variables nombradas dentro de una estructura se denominan “miembros”.

Un miembro, un rótulo y una variable ordinaria, esto es, no miembro, pueden tener el mismo nombre sin causar ningún conflicto, ya que se pueden distinguir fácilmente por el contexto en que se utilicen.

En diferentes estructuras se pueden encontrar los mismos nombres de miembros, sin que se tenga algún problema, pero por claridad y estilo se deberían utilizar los mismos nombres solo para objetos estrechamente relacionados.

Una declaración struct define un tipo, la llave derecha que cierra la lista de miembros puede ser seguida por una lista de variables, como lo hacemos para cualquier tipo básico:

```
struct {…..} x, y, z;
```

Es muy similar a escribir:

```
int x, y, z;
```

5.2.2 Estructuras y funciones

Las estructuras, como grupos de variables conectados lógicamente, se pueden pasar de una manera sencilla a las funciones, y las funciones pueden regresarlas.

Las únicas operaciones válidas sobre una estructura son copiarla o asignarla como unidad, tomar su dirección con & y tener acceso a sus miembros. La copia y la asignación incluyen pasarlas como argumentos a funciones y también regresar valores de funciones.

5.2.3 Arreglos de estructuras

La utilización más común de las estructuras puede ser la de “arreglo de estructuras”, primero hay que definir una estructura y luego declarar una variable tipo arreglo que contenga dicha estructura por ejemplo, declaremos la siguiente estructura:

```
struct dir
        char nombre [30];
        char calle [40];
        char ciudad [20];
        char prov [20];
```

```
        unsigned long int DP,
} ainfo, binfo, cinfo;
```

Con esto definimos una estructura llamada dir y se declaran las variables ainfo, binfo, cinfo, del tipo dir.

Ahora declararemos un arreglo de cien elementos de estructuras del tipo dir, esto es:

```
struct dir ainfo[100];
```

Esto crea cien conjuntos de variables que están organizadas como se definió en la estructura dir.

Para imprimir el código postal “DP” de la estructura 3 debemos escribir:

```
printf(“%d”,ainfo[2].DP);
```

Como en todas las variables de arreglos, los arreglos de estructura empiezan sus contadores en cero.

5.2.4 Apuntadores a estructuras

Para ejemplificar algunas consideraciones involucradas con apuntadores y arreglos de estructuras, tomaremos el siguiente ejemplo de un programa de conteo de palabras reservadas[6]:

```
#include <Stdio.h>
#include <ctype.h>
#include <string.h>
#define MAXWORD 100
```

[6] Ejemplo tomado del libro: El Lenguaje de Programación “C”, de Brian W. Kernighan y Dennis M. Ritchie, editorial Pearson Educación.

```
int getword(char *,int);
struct key *binsearch(char *. Struct key *, int);

/* cuenta palabras reservadas de C, con apuntadores */

main()
{
        char Word[MAXWORD];
        struct key *p;

        while (getword(Word,MAXWORD) != EOF)
                If (isalpha(Word[0]))
                        if((p=binsearch(Word, keytab, NKEYS)) != NULL)
                                p->count++;
        for(p = keytab; < keytab + NKEYS; p++)
                if (p-> count > 0)
                        printf("%4d %s/n",p->count, p->Word);
        return 0;
}
/*binsearch: encuantra una palabra en tab[0]… tab[n-1] */
struct key *binsearch(char *Word, struct key *tab, int n)
{

        int cond;
        struc key *low = &tab[0];
        struct key *high = &tab[n];
        struct key *mid;

        while (low < high){
                mid = low + (high-low) / 2;
                if((cond = strcmp(Word, mid->Word)) <0)
                        high = mid;
                else if(cond >0)
```

```
                low = mid +1;
        else
                return mid;
    }

    return NULL;
}
```

Este programa que cuenta palabras reservadas comienza con la definición de **keytab**. La función principal lee de la entrada con llamadas repetidas a la función getword, que trae una palabra a la vez. Cada palabra se consulta en keytab.

En este ejemplo debemos notar:

La declaración de **binsearch** debe indicar que regresa un apuntador a struct key en lugar de un entero, esto se declara en el prototipo de la función como en binsearch. Si la función binsearch encuentra la palabra, regresa el apuntador a ella, en caso contrario regresa NULL.

Ahora se tiene acceso a los elementos de keytab por medio de apuntadores.

Los inicializadores para low y high son apuntadores al inicio y justo después del final de la tabla.

Como la suma de apuntadores es ilegal, utilizaremos la resta, high-low es el número de elementos, por lo que mid = low + (high-low) /2 hará que mid apunte al elemento que está a la mitad entre low y high.

Este ejemplo quedará más claro cuando se halla estudiado el capítulo de apuntadores.

ACTIVIDAD DE APRENDIZAJE

El estudiante analizará en clase el ejemplo anterior, hasta comprender en su totalidad el funcionamiento de todo el programa.

El profesor deberá aclarar todas las dudas que se presenten.

5.2.5 Estructuras autorreferenciadas.

Para ejemplificar este concepto debemos resolver la siguiente pregunta: ¿Cómo podemos organizar los datos para manejar eficientemente una lista de palabras arbitrarias?, es decir minimizando el tiempo de ejecución de un programa.

Una posible solución es mantener siempre ordenado el conjunto de palabras que ya se han visto, colocando cada una en su posición correcta cuando llega. No se podría realizar recorriendo las palabras en un arreglo lineal, ya que el tiempo de ejecución sería demasiado, en lugar de esto se debe utilizar una estructura de datos llamada “árbol binario”.

Este árbol contiene un nodo por cada palabra distinta, cada uno de los nodos debe contener:

- Un apuntador al texto de la palabra.
- Una cuenta del número de ocurrencias.
- Un apuntador al nodo hijo de la izquierda.
- Un apuntador al nodo hijo de la derecha.

Un nodo solo puede tener 0, 1 o 2 hijos.

Cada uno de los nodos se representa convenientemente como una estructura con cuatro componentes:

```
struct tnode{                                   /* el nodo del árbol */
```

```
        char *word;                 /* apuntador al texto */
        int count;                  /* número de ocurrencias */
        struct tnode *left;         /* hijo a la izquierda */
        struct tnode *right;        /*/hijo a la derecha */
};
```

Esta declaración es recursiva, puede parecer riesgosa, pero es totalmente correcta.

Es inválido que una estructura contenga una instancia de sí misma, pero en el ejemplo:

```
struct tnode *left;
```

O cualquiera otra del ejemplo, declaran a left como un apuntador a tnode, no como un tnode en sí.

Esto es lo que se conoce como una estructura autorreferenciada, en ocasiones se requiere de una variante de este tipo de estructuras, en donde dos estructuras hacen referencia una a la otra.

ACTIVIDAD DE APRENDIZAJE

El estudiante del libro “El lenguaje de programación C” de Brian W Kernighan y Dennis M.Ritchie, editorial Pearson Educación, realizará, analizará, compilará y ejecutará los problemas presentados en las hojas 155 y 156 para entender el concepto de estructuras autorreferenciadas.

El profesor deberá aclarar todas las dudas que se presenten.

5.3 UNIONES

En el lenguaje de programación “C” una unión es un lugar de la memoria que se utiliza por algunas variables diferentes y posiblemente de diferentes tipos.

En el siguiente ejemplo se da la unión llamada u entre un carácter y un entero:

```
unión u {
        int i;
        char ca;
};
```

Como ocurre en el caso de las estructuras, esta unión no declara ninguna variable, la variable puede ser declarada colocando su nombre al final de la definición o por medio de una sentencia de declaración separada, para declarar una variable tipo unión llamada cnvt, utilizando la definición que acaba de darse se debe escribir:

```
union u cnvt;
```

5.4 CAMPOS DE BITS.

A diferencia de otros lenguajes “C” incluye un método para acceder a un solo bit

dentro de un byte método para acceder a un solo bit dentro de un byte, podemos mencionar como las utilidades principales de esta función:

- Si la capacidad del almacenaje es limitada se pueden almacenar algunas variables booleanas (cierto/falso) en un solo byte de información.
- Algunas interfaces de dispositivos transmiten información codificada en bytes dentro de un byte.
- Algunas rutinas criptográficas necesitan acceder a los bits en forma independiente dentro de un byte.

El método utilizado en ”C” para acceder a los bits está basado en las estructuras, como ejemplo mencionaremos la siguiente estructura donde se definen tres variables de un bit cada una:

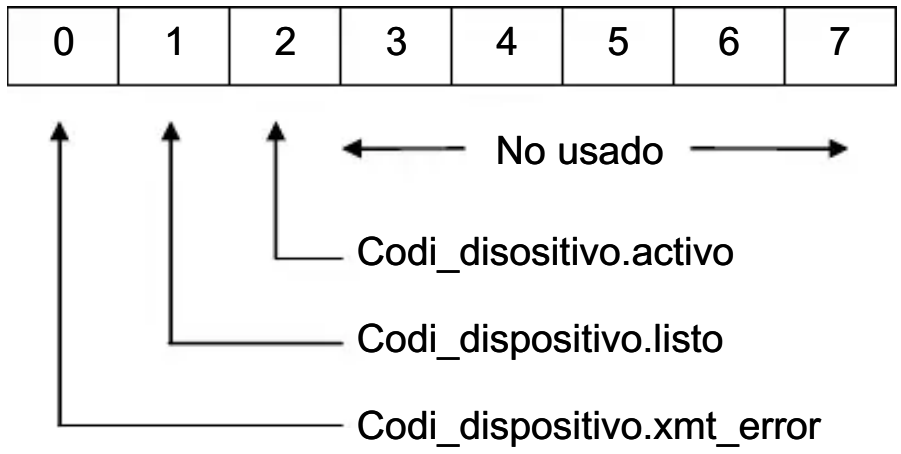
```
Struct dispositivo {
        Unsigned activo : 1;
        Unsigned listo : 1;
        Unsigned xmt_error : 1;
} codi_dispositivo;
```

Las tres variables se declaran como unsigned por que un bit no puede tener signo, los únicos valores que puede tener un bit son 0 y 1. La variable estructura codi_dispositivo se puede utilizar para decodificar información desde el puerto de una cinta magnética, por ejemplo el siguiente fragmento de código lo utilizamos para escribir un byte de información en la cinta y comprobaremos los errores utilizando codi_dispositivo.

```
Wr_cinta(c)
Char c;
{
        while(!codi_dispositivo-ready) rd(&codi_dispositivo); /* espera */
        wr_to_tape(c); /* escribe un byte */

        while (codi_dispositivo.activo) rd(&codi_dispositivo); /* espera hasta que
                                                la  información se escriba */
        if(dev_code.xmt_error) printf(“error de escritura”);
}
```

A continuación escribiremos la variable del campo de bit codi_dispositivo como se representa en memoria:


0
1
2
3
4
5
6
7
No usado
Codi_disositivo.activo
Codi_dispositivo.listo
Codi_dispositivo.xmt_error

AUTOEVALUACIÓN

1. ¿Qué es un arreglo?

    - En “C” los arreglos se componen de posiciones contiguas de memoria, donde la dirección más baja corresponde al primer elemento y la más alta al último elemento.

2. ¿En “C” existen arreglos multidimensionales?

    - *Cierto*

3. ¿Qué es una estructura?

    - *Una estructura en “C” es una colección de una o más variables, de tipos posiblemente diferentes, que se referencian bajo un mismo nombre, con el objeto de mantener en un sitio la información que está relacionada..*

4. Defina una estructura denominada **jugador** que sea capaz de almacenar la siguiente información: nombre del jugador, nombre del grupo, tanteo medio.

```
struct jugador{
        char jugador_nombre[40];
        char jugador_grupo[40];
        float tanteo_medio;
};
```

5. Utilizando **jugador** de la pregunta anterior, declarar un arreglo de estructuras de 100 elementos. Llamarlo p_info.

```
struct jugador p_info[100]
```

UNIDAD 6

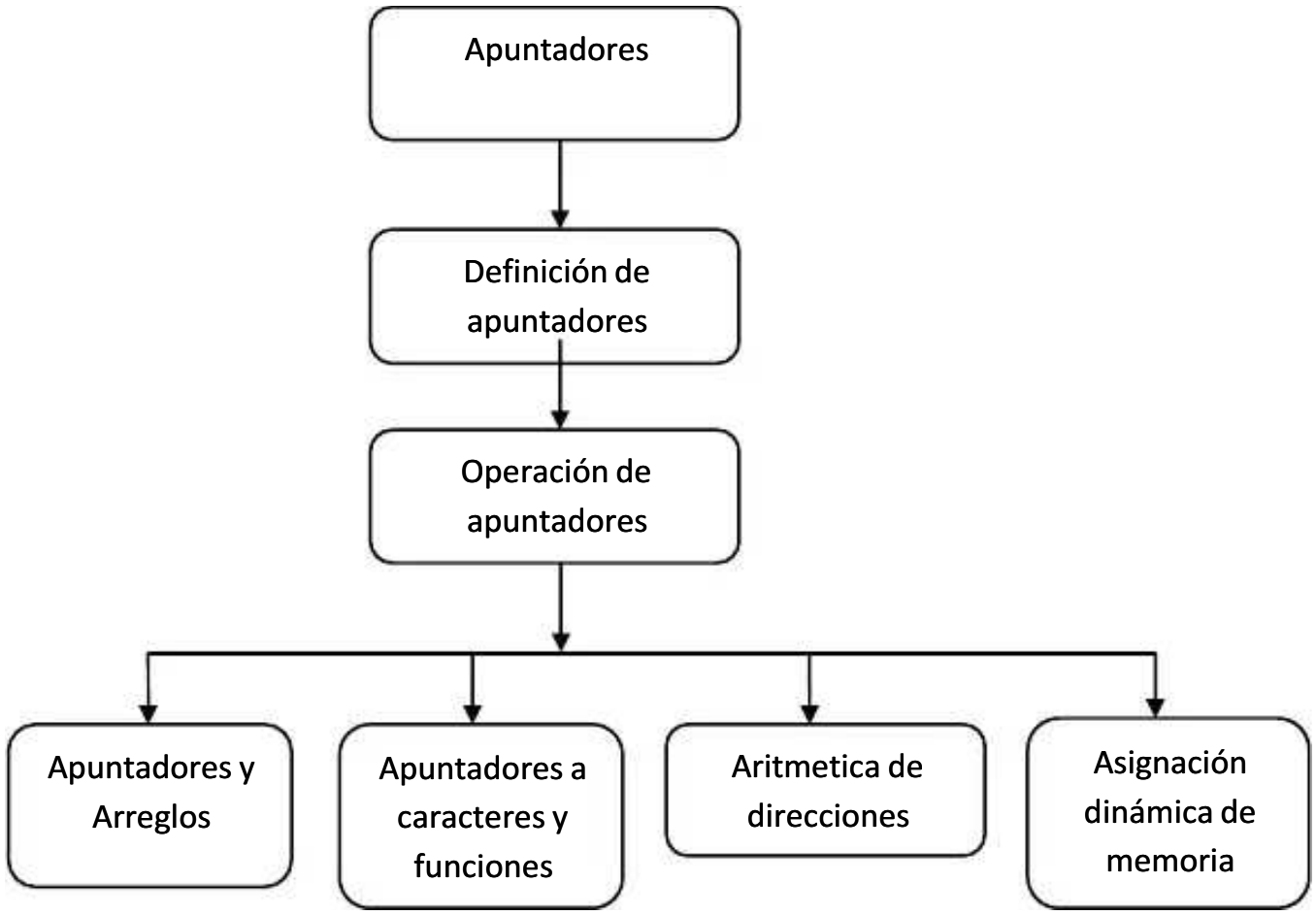
APUNTADORES

OBJETIVO:

Que el estudiante aprenda el concepto de apuntadores, las operaciones que pueden realizarse con ellos, comprenda la aritmética de direcciones para utilizarlos en la solución de problemas complejos de programación.

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

En capítulos anteriores ya se han utilizado los apuntadores, pero hasta este capítulo aprenderemos a utilizarlos con todo detalle, esto nos permitirá realizar programas cuyo tiempo de ejecución sea más rápido.

El uso de los apuntadores es importante para realizar de forma correcta programas en el lenguaje de programación “C”.

El alumno aprenderá:

- El concepto y manejo de arreglos y estructuras en el lenguaje de programación “C”
- Aprenderá el manejo de uniones y el concepto de campos de bits herramientas importantes en “C”.

6.1 DEFINICIÓN DE APUNTADORES.

En pocas palabras un apuntador es una variable que contiene una dirección de memoria. Esta dirección es la posición de alguna variable en memoria.

Entonces podemos decir que un apuntador “apunta” a una variable, a la que se puede accesar de forma indirecta con los operadores especiales sobre punteros: * y **&.**

Una computadora tiene un arreglo de celdas de memoria numeradas o direccionadas consecutivamente, esta celdas se pueden manipular individualmente o en grupos contiguos.

Cualquier byte puede ser un chat.

Un par de celdas de un byte pueden ser un entero short

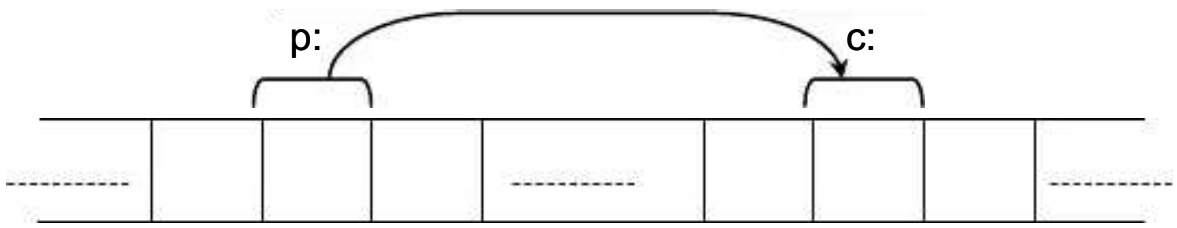
Cuatro bytes consecutivos forman un long.

Un apuntador es un grupo de celdas (generalmente 2 o 4) que pueden mantener una dirección.

Una computadora típica tiene un arreglo de celdas de memoria numeradas consecutivamente, que pueden manipularse individualmente o en grupos contiguos, recordemos que cualquier byte puede ser un char, un par de

celdas de un byte pueden tratarse como un entero short, y cuatro bytes contiguos forman un long, siempre un apuntador es un grupo de celdas que pueden mantener una dirección.

A manera de ejemplo vamos a considerar: c es una variable tipo chat, p es un apuntador, que está apuntando a la variable c, este ejemplo puede representarse como se muestra en la siguiente figura:



6.2 Operación de Apuntadores

Existen dos operadores especiales sobre apuntadores & y *, ambos son operadores unarios, Recordemos que un operador unario se aplica a un solo operando.

El operador **&** sirve para dar la dirección de un objeto, y solo se aplica a objetos que están en memoria, como son variables y elementos de arreglos, no puede aplicarse a expresiones, constantes o variables tipo registro, lo podemos recordar como “la dirección de”,

La proposición:

p = &n;

Asigna la dirección de n a la variable p, por esto podemos decir que:

p “apunta a“ n

El operador ***** que se conoce como operador de indirección o desreferencia, accede al contenido de una variable cuya dirección es el valor de un puntero, lo podemos recordar cómo “en la dirección”.

Por ejemplo si **xyz** y **k** son variables enteras y **h** es un apuntador a entero, entonces:

h = &xyz;

k = *h;

asigna el valor de xyz a k

Los apuntadores tienen que haber sido declarados.

Para declarar h como un apuntador a entero, debemos utilizar:

int *h;

Para declarar x como un apuntador a float, se indica:

```
float *x;
```

Debemos de tener la seguridad de que las variables tipo apuntador siempre apunten al tipo de datos correcto, cuando declaramos que un apuntador es del tipo int, entonces cualquier dirección que tenga este apuntador debe apuntar a una variable entera.

Como sucede con cualquier variable, un apuntador se puede utilizar en el lado derecho de las sentencias de asignación para asignar su valor a otra variable, como lo podemos ver a continuación:

```
int x;
unsigned y;
int *p1, p2;
p1= &x;                 /* da la dirección de x a p2 */
y=p2;
printf (“%u”,y);        /* imprime el valor decimal de la dirección de x ¡no el
                           valor de x! */
```

6.3 Apuntadores y Arreglos

En el lenguaje de programación “C” existe una fuerte relación entre los apuntadores y los arreglos, Cualquier operación que pueda lograrse por medio de indexación de un arreglo también puede lograrse por apuntadores, con el uso de apuntadores se logra una versión de ejecución más rápida, pero algo más difícil de entender.

Los apuntadores de cualquier tipo de arreglos funcionan como una alternativa a la indexación.

La declaración:

```
int m[10];
```

Define un arreglo m de tamaño 10, es decir de 10 objetos consecutivos agrupados en memoria, llamados m[0], m[1], m[2], m[3],…..,m[9]

Cada uno de los 10 objetos es del tipo entero

La notación m[i] se refiere al (i-ésimo +1) elemento del arreglo, por ejemplo: m[3] se refuere al cuarto elemento del arreglo, ya que el primer índice es 0, el segundo 1, el tercero 2 y el cuarto elemento lleva el índice 3

Si pm es un apuntador a un entero y fue declarado como:

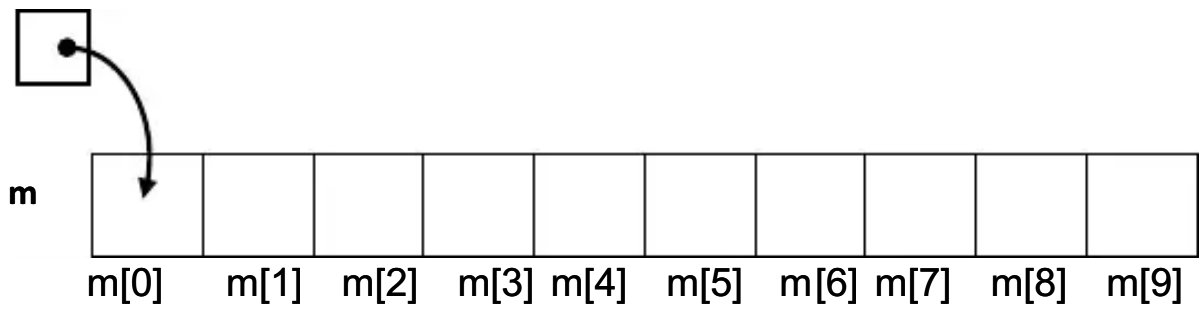
```
int *pm;
```

la asignación siguiente:

```
pm = &m[0];
```

hace que pm apunte al elemento cero del arreglo m; es decir pm contiene la dirección de m[0]

pm:



Ahora la siguiente asignación:

```
x=*pm;
```

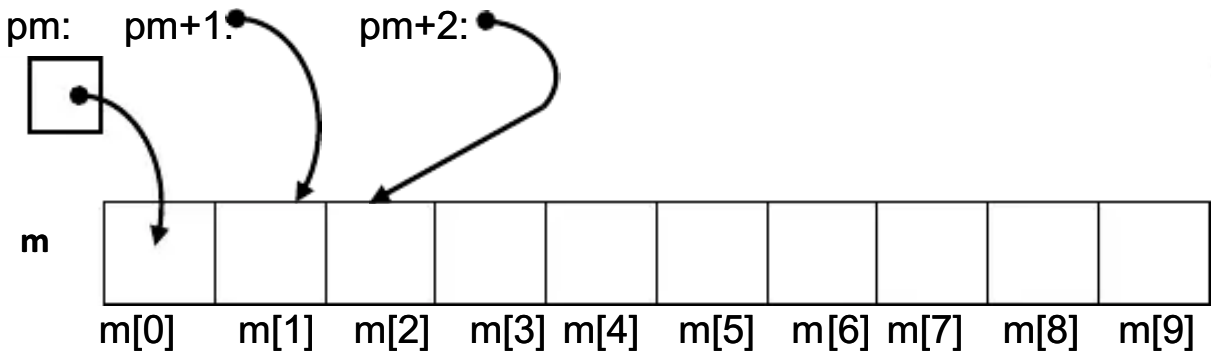
copiará el contenido de m[0] en la variable x.

Si pm apunta a un elemento en particular de un arreglo, entonces pm +1 apunta al siguiente elemento, pm + i apuntará i elementos después de pm, de manera inversa pm -1 apunta un elemento anterior a pm y pm – i apunta i elementos anteriores a pm.

Si pm apunta a m[0],

*(pm+1)

Se refiere al contenido de m[1], pm + i es la dirección de m[i] y *(pm+i) es el contenido de m[i]



Esto es correcto sin importar el tipo o tamaño de las variables del arreglo m,

La relación entre indexación y aritmética de apuntadores es muy estrecha.

ACTIVIDAD DE APRENDIZAJE

El estudiante realizará la solución de un problema definido por el profesor de dos maneras distintas:

Mediante el uso de arreglos con índices.

Mediante el uso de arreglos y apuntadores

Nota: En lo posible tratar de comparar los tiempos de ejecución de ambas soluciones.

6.4 ARITMÉTICA DE DIRECCIONES.

El significado de “agregar 1 a un apuntador”, y por extensión, toda la aritmética de apuntadores, es que pm+1 apunta al siguiente objeto delante de pm y pm+i apunta al i-ésimo objeto delante de pm.

Solo existen dos operadores aritméticos que se pueden utilizar con los apuntadores, + y -.

Para entender lo que ocurre con la aritmética de apuntadores, supongamos que p1 es un apuntador a entero con valor actual de 2000, despu♪0s de la expresión:

p1++;

El contenido de p1 será 2002 y no 2001. Cada vez que p1 se incrementa, apuntará al entero siguiente, que en casi todas las computadoras ocupan 2 bytes, esto es igual en los decrementos, por ejemplo:

p1--;

Hará que p1 tenga el valor de 1998, suponiendo que originalmente era 2000.

Debemos recordar que cada vez que un apuntador se incrementa apuntará a la posición de memoria del siguiente elemento de su tipo y cada vez que se decremente, apuntará a la posición del elemento anterior de su tipo.

Los apuntadores a carácter normalmente coinciden con la aritmética normal, pero para todos los demás apuntadores se incrementarán o decrementarán según la longitud del tipo de dato al que apunten

6.5 APUNTADORES A CARACTERES Y FUNCIONES

Una constante de caracteres, escrita como:

“Soy una cadena”

Al estar entre comillas “ ” se considera constante de caracteres.

Este conjunto de caracteres, en la representación interna, termina con un carácter nulo ‘\0’, de tal manera que los programas pueden encontrar el fin, la longitud de almacenamiento es el número de caracteres entre las comillas mas uno, que es el carácter nulo.

La más común ocurrencia de cadenas de caracteres constantes se encuentra como argumentos de funciones, por ejemplo;

printf(“hola, mundo\n”);

Cuando una cadena de caracteres, como en el ejemplo anterior, aparece en un programa, el acceso a dicha cadena es atraves de un apuntador a caracteres, la función printf recibe un apuntador al inicio de la cadena de caracteres, accesando de esta forma al primer elemento de la cadena de caracteres.

Las cadenas constantes no necesitan ser argumentos de funciones. Si declaramos una variable apuntador llamada pmensaje como:

char *pmensaje;

Entonces la proposición:

pmensaje = “hola”;

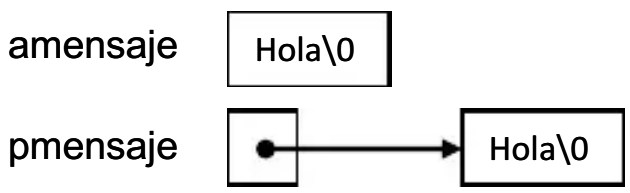
Se asigna a pmensaje una puntador al arreglo de caracteres, no es la copia de la cadena.

Para entenderlo mejor vamos analizar las siguientes definiciones:

char amensaje[]=”hola”; /* arreglo */

char *pmensaje=”hola” /* apuntador */

la primera expresión declara amensaje como un arreglo, con el tamaño suficiente para contener la cadena de caracteres mas el carácter nulo que lo finaliza '\0', podemos modificar caracteres individuales dentro del arreglo, amensaje, se referirá a la misma localidad de memoria. En cambio pmensaje es un apuntador, que se inicializó para apuntar a una cadena constante, posteriormente el apuntador puede modificarse para que apunte a cualquier otra posición de memoria.



En el lenguaje de programación "C" una función por sí sola no es una variable, pero se pueden definir apuntadores a funciones, que pueden asignarse, colocarse en arreglos, pasados y regresados a funciones, etc., esto debido a que una función si bien no es una variable sigue teniendo una posición en memoria. A continuación este apuntador puede ser útil para las llamadas a funciones.

ACTIVIDAD DE APRENDIZAJE

El estudiante realizará y analizará el siguiente ejemplo tomado del libro Programación en lenguaje C del autor Herbert Schildt, editorial Mc Graw Hill:

```
main()
{
        int strcmp();          /*declara una función */
        char s1[80, s2[80];
        gets(s1);
        gets(s2);
```

```
        comp(s1,s2,strcmp);
}
Comp(a,b,cmp)
Char *a,*b;
Int (*cmp) ();
{
        printf("comprueba la igualdad\n");
        if (!(*cmp) (a,b)) printf("igual");
        else printf("distinto");
}
```

Este programa utiliza un puntero a función, y es na función estándar de comparación de cadenas que se encuentra en la librería de "C". Se declara en main(), cuando se llama a la función comp() se pasan como parámetros dos caracteres y un puntero a función.

6.6 Asignación dinámica de memoria

Diseñar un programa de computadora se puede comparar a diseñar un edificio, debemos tener presentes diversas consideraciones funcionales y estéticas que contribuyen al resultado final. Por ejemplo, algunos programas son funcionalmente rígidos, como una casa, con un cierto número de dormitorios, una cocina, dos baños, etc. Otros programas tienen que ser abiertos como centros de convenciones, con paredes móviles y revestimientos modulares que las permitan adaptarse a diversas necesidades. En este capítulo presentamos los mecanismos de almacenamiento que permiten escribir programas flexibles[7].

[7] Nota para una mayor claridad, este tema se tomó del libro Programación Avanzada del Lenguaje "C", de Herbert Schildt, ya que se explica de manera muy concisa.

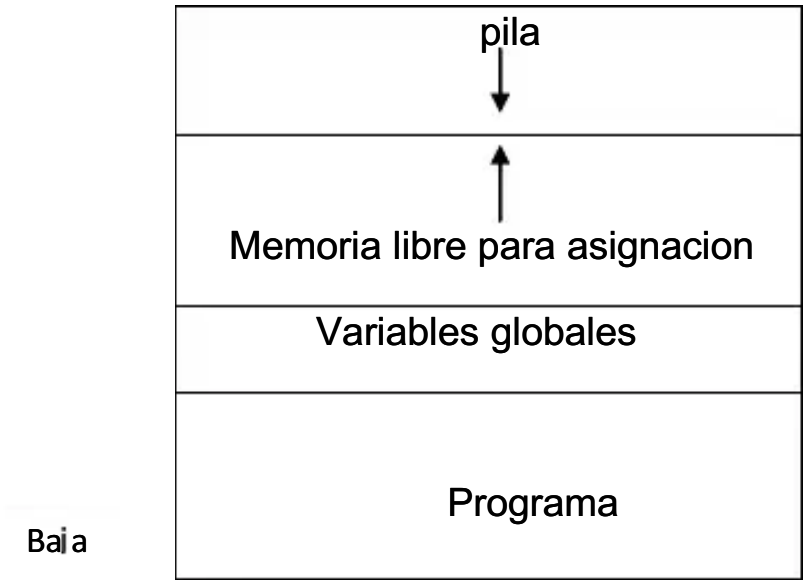
Existen dos formas en las que un programa escrito en lenguaje “C” puede almacenar información en la memoria principal de la computadora. La primera utiliza variables locales y globales –incluyendo arreglos y estructuras- que son definidas por el lenguaje C. Para las variables globales, el almacenamiento es fijo durante todo el tiempo de ejecución del programa. Para las variables locales, el almacenamiento se sitúa en el espacio de la pila de la computadora. Aunque las variables globales y locales están eficientemente implementadas en C, requieren que el programador conozca de antemano la cantidad de almacenamiento necesaria en cada situación

La segunda forma, y la más eficiente, en la que se puede almacenar la información es utilizando las funciones de asignación dinámica de memoria del lenguaje “C” **malloc()** y **free()**.

El almacenamiento de la información se realiza en el área de memoria libre se queda entre el área de almacenamiento permanente del programa y la pila (que es utilizada por C para almacenar las variables locales). La figura siguiente muestra cómo un programa en C aparecería en memoria. La pila cuando se utiliza crece hacia abajo, de forma que la cantidad de memoria necesaria está determinada por la forma en que está diseñado el programa. Por ejemplo, un programa con muchas funciones recursivas hace mucho más grande la demanda de memoria en pila que un programa que no tiene funciones recursivas, debido a que las variables locales se almacenan en la pila. La memoria necesaria para un programa y los datos globales es fija durante la ejecución del programa. La memoria para satisfacer una petición de **malloc()** se obtiene del área de memoria libre, que comienza encima de las variables globales y crece hacia la pila. Bajo casos bastantes extremos es posible que la pila se introduzca en la memoria asignada.

Memoria del sistema

Alta



La utilización de la memoria de un programa en C

Si usted no está familiarizado a fondo con **malloc()** y **free()**, aquí hay una pequeña revisión.

Una revisión de **malloc()** y **free()**

Las funciones **malloc()** y **free()** forman el sistema de asignación dinámica de memoria del lenguaje “C” y son parte de la librería estándar de este lenguaje. Funcionan juntas utilizando la región de memoria libre que queda entre el programa y el principio de la pila para establecer y mantener una lista de almacenamiento disponible. Cada vez que se hace una petición de memoria por **malloc()**, se asigna una porción de la memoria libre restante. Cada vez que se hace una llamada para liberar memoria con **free()** es organizar la memoria en una lista enlazada.

La función **malloc()** es la función de asignación dinámica de memoria de propósito general de “C”. La forma general de llamada es

```
char *malloc(), *p;
int número_bytes;
p=malloc(número_bytes);
```

P almacena un puntero al primer byte de la región de memoria. Si no hay bastante memoria disponible para satisfacer la petición de **malloc()**, ocurre un fallo de asignación y **malloc()** devuelve el valor cero. La función **malloc()** siempre necesita saber el número de bytes que tiene que asignar -incluso si la información que se necesita almacenar es algún otro tipo de datos, tal como un entero o una estructura-. Se puede utilizar sizeof para determinar el número exacto de bytes necesarios para cada tipo de datos. Esto ayuda a hacer transportables sus programas a una variedad de sistemas. Incluso aunque se devuelve un puntero a un carácter, se puede asignar a un puntero del tipo correcto para satisfacer cualquier necesidad de programación Antes de utilizar el puntero devuelto por **malloc()** siempre asegúrese de que la asignación pedida tuvo éxito comparando el valor devuelto con 0. No utilice un puntero de valor 0, probablemente colgará el sistema.

La función **free()** es la contraria a **malloc()**: devuelve al sistema la memoria previamente asignada. Esta parte de la memoria se puede volver a utilizar en subsiguientes llamadas a **malloc()**. La forma general de **free()** es

```
char *p;  /* asume que p tiene un puntero válido */
free(p);
```

Recuerde que nunca se debe llamar a **free()** con un argumento erróneo, ya que la lista de memoria libre se podría destruir.

El siguiente programa asigna el almacenamiento necesario para 40 enteros, imprime sus valores y libera la memoria al sistema. Aquí sizeof se utiliza para asegurar la transportabilidad a otros tipos de computadoras.

```
main ()       /* un corto ejemplo de asignación */
{
        int *p, t;
        p=malloc(40*sizeof(int));

        if(p==0) {
                printf("no hay memoria\n");
```

```
                exit (0);
}
        for (t=0;t<40;++t) *(p+t)=t;
        for (t=0;t<40;++t) printf ("%d ",*(p+t));
        free(p9);
```

AUTOEVALUACIÓN

1. ¿Qué es un apuntador?

   Es una variable que contiene una dirección de memoria. Esta dirección es la posición de alguna variable en memoria.

2. Suponiendo que n es un entero y p un apuntador a un entero, escribir el fragmento de código que asignará el valor de 10 a n utilizando el apuntador p.

   ```
   p = &n;
   *p = 10;
   ```

3. ¿Qué operadores aritméticos pueden usarse con los apuntadores?

   Solo existen dos operadores aritméticos que se pueden utilizar con los apuntadores, + y -.

4. Qué función tienen malloc() y free() en “C”.

   forman el sistema de asignación dinámica de memoria del lenguaje “C” y son parte de la librería estándar de este lenguaje, malloc() hace una petición de memoria y free() es una llamada para liberar memoria.

5. Mencione los operadores especiales sobre apuntadores, y explique su funcionamiento.

   *Existen dos operadores especiales sobre apuntadores & y *, ambos son operadores unarios.*

   *El operador & sirve para dar la dirección de un objeto, y solo se aplica a objetos que están en memoria (variables y elementos de arreglos) El operador * accede al contenido de una variable cuya dirección es el valor de un puntero.*

UNIDAD 7

ARCHIVOS Y ENTRADA / SALIDA

OBJETIVO:

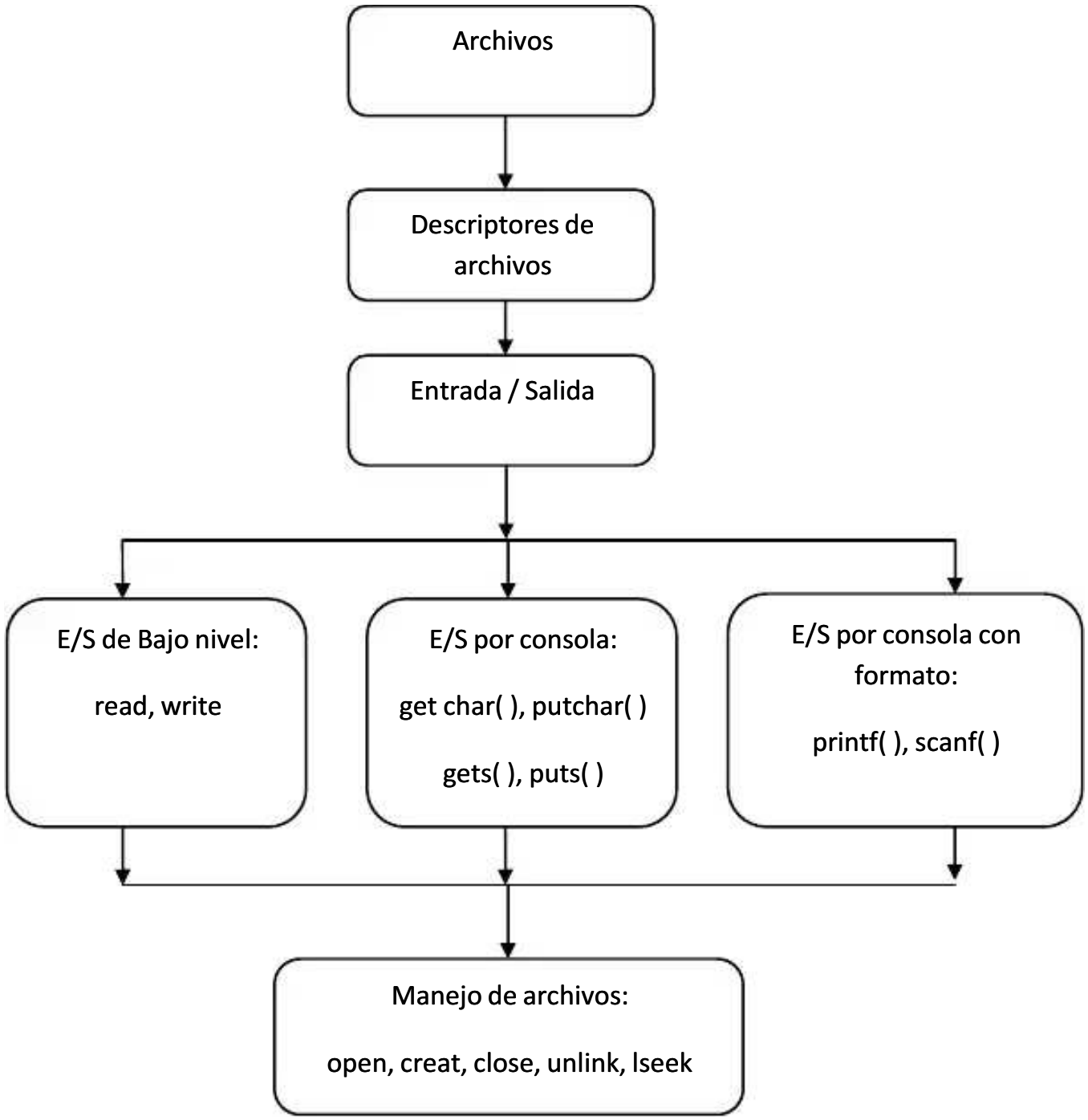
El estudiante aprenderá:

Qué son y cómo funcionan los archivos, así como las principales funciones utilizadas para su manejo.

Las diferentes funciones de entrada / salida existentes en el lenguaje de programación “C”

TEMARIO

MAPA CONCEPTUAL



INTRODUCCIÓN

El manejo de archivos en “C” y las funciones de entrada / salida tienen vital importancia para poder intercambiar información con los programas, funcionan como interfaz entre los programas y el usuario, se analizarán diferentes funciones de entrada / salida, aprenderemos los descriptores de archivo y cómo redireccionar las entradas y salidas estándar hacia archivos u otros dispositivos.

El alumno aprenderá:

- Cómo deben manejarse los archivos en “C”
- Las principales funciones de Entrada / Salida que existen en “C”.

7.1 DESCRIPTORES DE ARCHIVOS

Todas las entradas y salidas de datos se realizan por medio de lectura o escritura de archivos, incluso los dispositivos periféricos como son el teclado y la pantalla son vistos como archivos que están en el sistema. Existe una sencilla interfaz que maneja todas las comunicaciones entre un programa y los dispositivos periféricos.

Antes de leer o escribir en archivo es necesario informarle al sistema nuestra intención de hacerlo, mediante el proceso de abrir un archivo, si se va a escribir en un archivo, también puede ser necesario crear dicho archivo o eliminar el contenido previo. El sistema verifica si se cuenta con los derechos suficientes para realizar dichas acciones, por ejemplo:

¿El archivo existe?

¿Se cuenta con permisos para tener acceso al archivo?

Si todo es correcto se regresa al programa un entero pequeño no negativo llamado descriptor de archivo. Siempre que vamos a realizar operaciones de entrada / salida sobre dicho archivo se utiliza el descriptor de archivo como identificación en lugar del nombre de archivo, un descriptor de archivo es similar al apuntador de archivo usado por la biblioteca estándar.

Toda la información referente a un archivo abierto es mantenida por el sistema, el programa de un usuario hace referencia a los archivos solo por el descriptor

Debido a que es muy frecuente que tanto la entrada como la salida se refieran al teclado y a la pantalla, en “C” existen arreglos especiales para hacer esto de manera conveniente, cuando el “Shell” o intérprete de comandos ejecuta cualquier programa abre tres archivos con descriptores 0, 1 y 2, que son llamados “entrada estándar”, “salida estándar” y “error estándar” respectivamente, por ejemplo si un programa lee de 0 y escribe en 1 o en 2, puede hacer operaciones de entrada / salida sin preocuparse de abrir archivos.

Para redirigir la entrada / salida estándar hacia o desde un archivo usando los símbolos: **<** y **>**.

prog <archivodeentrada >archivodesalida

En el ejemplo anterior el intérprete de comandos cambia las asignaciones predefinidas para los descriptores 0 y 1 a los archivos indicados, en la mayoría de los programas dejamos el descriptor de archivo 2 asignado a la pantalla, para poder visualizar rápidamente los mensajes de error en la pantalla de la computadora. La asignación de archivos la cambia el Shell o interprete de comandos, no el programa.

7.2 E/S DE BAJO NIVEL: READ Y WRITE.

Desde programas escritos en “C’ se tiene acceso a la entrada y salida por medio de las funciones read y write, para las cuáles el primer argumento es un descriptor de archivo, el segundo argumento es un arreglo de caracteres perteneciente al programa hacia o de donde los datos van a ir o a venir y el tercer argumento es el número de bytes que serán transmitidos, a continuación tenemos dos ejemplos:

int n_leidos = read(int fd, char *buf, int n);

int n_escritos = write(int fd, char *buf, int n);

Cada una de las llamadas regresa una cuenta del número de bytes transferidos, en el proceso de lectura el número de bytes regresados puede ser menor al número solicitado, un valor de regreso igual a cero bytes implica fin de archivo y -1 indica algún tipo de error.

Para la escritura, el valor de retorno es el número debytes escritos, si este no es igual al número solicitado ha ocurrido algún error.

En una lectura pueden leerse cualquier cantidad de bytes, sin embargo los valores mas comúnes son:

1 significa carácter por carácter a la vez, sin buffer

1024 o 4098 que corresponden al tamaño de un bloque físico de un dispositivo periférico, resultando más eficientes los valores más grandes debido a que serán realizadas menos llamadas al sistema.

Actividad de aprendizaje

El estudiante analizará y discutirá en clase el siguiente problema tomado del libro “el lenguaje de programación “C”, Brian W. Kernighan y Dennis M. Ritchie:

```
#include “syscall.h”

main()                          /* copia la entrada a la salida */
{
        char buf(BUFSIZ);
        int n;

        while ((n=read(0, buf, BUFSIZ)) > 0)
        write(1, buf, n);
        return 0;
}
```

Se han reunido prototipos de funciones para las llamadas al sistema en un archivo llamado syscall.h, pero este nombre no es estándar.
El parámetro BUFSIZ está definido dentro de syscall.h; su valor es de un tamaño adecuado para el sistema en particular que se utilice

---

7.3 E/S POR CONSOLA: GETCHAR( ) Y PUTCHAR( ), GETS( ) Y PUTS( ).

Las funciones de entrada / salida “E/S” por consola más simples son:

getchar() lee un carácter desde la entrada estándar, normalmente asignada al teclado. Espera a que se pulse una tecla y después devuelve su valor, normalmente envía el "eco" del carácter pulsado a la pantalla, es decir que el carácter que se oprime en el teclado se mostrará en la pantalla sin necesidad de indicarlo específicamente.

putchar() imprime un carácter en la salida estándar, normalmente asignada a la pantalla. Escribe el argumento un carácter en la pantalla de la computadora, pero solamente si dicho argumento es parte del conjunto de caracteres que la computadora puede visualizar.

ACTIVIDAD DE APRENDIZAJE

El estudiante realizará el siguiente programa el cuál toma caracteres desde el teclado (entrada estándar) y los mostrará en la pantalla (salida estándar) cambiando las mayúsculas por minúsculas y viceversa, para finalizar el programa pulsar un punto "**.**" La función islower() devuelve "cierto" si **ch** (carácter pulsado en teclado) es un carácter en minúscula. La función toupper() convierte una letra minúscula en mayúscula y la función tolower() convertirá una letra mayúscula en minúscula, ninguna de estas dos funciones afecta a los caracteres especiales como +, ¿, %, etc. Estas funciones se encuentran en la librería estándar de "C".

```
main ()        /* cambia tipo de letra mayúsculas por minúsculas y viceversa */
        {

                char ch;

                do {

                        ch=getchar();

                        if(islower(ch)) putchar(toupper(ch));
```

```
                else putchar(tolower(ch));
        } while (ch!='.');        /* use un punto para parar */
}
```

Las funciones que nos permiten leer cadenas de caracteres por consola, son el siguiente paso en complejidad y potencia son:

Gets() devuelve una cadena terminada con el carácter nulo en el arreglo de caracteres que recibe como argumento, se pueden teclear caracteres desde el teclado hasta que se pulse enter, con esto se coloca un carácter nulo al final de la cadena y gets() devuelve el valor. El enter o retorno de carro no va en la cadena, por lo que esta función no se puede utilizar para devolver el retorno de carro, sin embargo getchar() si lo puede hacer. Gets() permite corregir errores de escritura por medio de la tecla “backspace” antes de pulsar enter o retorno de carro.

Puts() escribe su argumento de cadena en la pantalla, reconoce los mismos códigos de barra invertida que printf(), como puede ser \n para cambio de línea. Una llamada a puts() produce menos sobrecarga al procesador que la llamada a printf(), debido a que solo saca a pantalla una cadena de caracteres, no puede sacar números o hacer conversiones de formatos, por esto ocupa menos espacio y es mas rápida que printf() cuando muestra cadenas en la pantalla, está en la mayoría de las librerías estándar

7.4 E/S POR CONSOLA CON FORMATO PRINTF( ) Y SCANF( ).

Además de las funciones sencillas de E/S mostradas en los temas anteriores, la librería estándar de “C” incluye dos funciones que permiten realizar entrada y

salida formateada, la E/S formateada se refiere al hecho de que con estas funciones se puede formatear la información, recordemos que las funciones descritas anteriormente solo permiten introducir o mostrar los datos en forma de filas, estas funciones son pritf() y scanf().

printf()

El formato general de esta función es:

printf(“cadena de control”, lista de argumentos);

La cadena de control contiene dos tipos de elementos: el primero está formado por caracteres que se imprimirán en la pantalla, el segundo tipo contiene los comandos de formato, que nos sirven para definir la forma en que se mostrarán en pantalla los siguientes argumentos, debe existir exactamente el mismo número de comandos de formato que de argumentos, los comandos de formato y los argumentos se aparean por orden.

Por ejemplo:

printf (“Hi %c %d %s”, ‘c’,10,”allí!”);

Mostrará en pantalla:

Hi c 10 allí!

La siguiente tabla muestra los códigos de control de formato de la función printf():

%c	Un único carácter.
%d	Decimal.
%e	Notación científica.
%f	Coma flotante decimal.
%g	Utiliza %e o %f, la que sea mas corta.

%o	Octal.
%s	Cadena de caracteres.
%u	Decimal sin signo.
%x	Hexadecimal.

Estos códigos de control de formato pueden tener modificadores, por ejemplo que especifiquen la anchura del campo, el número de decimales, indicador de ajuste a la izquierda. Si colocamos un entero entre el % y el comando de formateo actuará como un especificador de anchura mínima del campo, por lo que se rellena la salida con espacios en blanco o con ceros, para asegunar la longitud mínima, si la cadena o el número son mayores que el número mínimo indicado se imprimirá entero aunque sobrepase el número mínimo indicado.

Por default siempre el relleno se hace con espacios en blanco, por lo que si requerimos rellenar con ceros se coloca un 0 delante del especificador de anchura del campo, por ejemplo: **%05d** rellenará con ceros los números con longitud inferior a los 5 dígitos, **%10.4f** mostrará un número de almenos 10 caracteres con cuatro decimales, %5.7s muestra una cadena de caracteres que

tendrá al menos cinco caracteres, pero que no excederá de siete.

scanf()

Esta es una rutina de entrada de propósito general, permite leer datos formateados y convertir automáticamente la información numérica a enteros o flotantes, su formato general es:

Scanf(cadena de control, lista de argumentos);

La cadena de control está formada por códigos de formatos de entrada, como se muestra en la siguiente tabla:

%c	Lee un único carácter.
%d	Lee un entero decimal.
%e	Lee un número en punto flotante.
%f	Lee un número en punto flotante.
%h	Lee un entero corto.
%o	Lee un número octal.
%s	Lee una cadena de caracteres.
%x	Lee un número hexadecimal.

Los comandos de formato pueden utilizar modificadores de la longitud de campo, como en el caso de printf() son enteros que se colocan entre el % y el código de comando de formato, un * colocado después del % suprimirá la asignación y avanzará a la siguiente entrada del campo.

Todas las variables utilizadas para recibir valores por medio de esta función se pasan por sus direcciones, por lo que todos los argumentos, distintos a los de la cadena de control, tienen que apuntar a las variables que recibirán la entrada, por ejemplo:

scanf(“%d”,&cuenta);

Lee un entero dentro de la variable cuenta.

Las cadenas de caracteres se leerán sobre arreglos de caracteres, por ejemplo para leer una cadena dentro del arreglo de caracteres dirección, utilizamos:

scanf(“%s”,dirección);

En este ejemplo dirección ya es un apuntador y no es necesarios que vaya precedido por el operador &.

Si se tiene cualquier otro carácter en la cadena de control, incluyendo espacios en blanco, tabuladores y cambios de línea, se utilizarán para comprobarlos con los caracteres del flujo de entrada, cualquier carácter que se aparee con ellos será descartado, ver el segundo ejemplo indicado a continuación.

Ejemplos de scanf()

Para el flujo de entrada: x y,

scanf(“%c%c%c”,&a,&b,&c);

asignará el carácter x a la variable a, un espacio en blanco a la variable b y el carácter y en la variable c.

Para el flujo de entrada abcdttttttefg,

scanf(“st%s”,nombre1,nombre2);

asignará los caracteres abcd en nombre1 y los caracteres efg en nombre2, las letras “t”

ACTIVIDAD DE APRENDIZAJE

El estudiante realizará al menos 10 ejemplos más de formatos con la función printf() y 10 más con la función scanf()

7.5 Manejo de archivos.

A continuación se describen las funciones principales existentes en “C” para el manejo de archivos.

7.5.1 Open

Esta función regresa un descriptor de archivo que es tan solo un int, regresa -1 si ocurrea lgún error.

```
include <fcntl.h>
int fd;
int open(char *nombre, int flags, int perms);
fd=open(nombre,flags,perms);
```

El argumento nombre es una cadena de caracteres que contiene el nombre del archivo, el segundo argumento (flags) es un int que especifica como se debe abrir el archivo:

O_READONLY abrir solo para lectura.

O_WRONLY abrir solo para escritura.

O_RDWR abrir tanto para lectura como para escritura.

Estas constantes están definidas en <fcnl.h> cuando usamos, por ejemplo UNIX System V, y en <sys/file.h> en otros sistemas, por ejemplo Unix con Berkeley (BSD).

Ejemplo:

Abrir un archivo ya existente para lectura:

```
fd = open(nombre, O_RDONLY,0);
```

7.5.2 Creat

Sería un error tratar de abrir un archivo que no exista, con la instrucción de arriba, para poder crear nuevos archivos o reescribir archivos viejos, se utiliza la llamada al sistema creat:

```
int creat(char *nombre,int perms);
```

fd = creat(nombre, perms);

Esta función regresa un descriptor de archivo si pudo crear el archivo, y un -1 si no pudo ser creado.

Si el archivo ya existe, esta función lo truncará a longitud cero, descartando su contenido previo, no significa un error crear con creat un archivo existente, en caso de que no exista creat lo crea con los permisos que especificamos con el argumento perms.

En sistemas de archivos con el SO UNIX hay 9 bits para información de permisos asociados con archivos, que nos permiten controlar el acceso de lectura, escritura y ejecución tanto para el propietario del archivo, para el grupo del propietario y para el resto de los usuarios que tengan acceso al archivo. Por esto un número octal de 3 dígitos es apropiado para especificar los permisos de acceso a un archivo, por ejemplo 0755, especifica permisos para leer, escribir y ejecutar solo para el propietario del archivo y de lectura y ejecución para el grupo al que pertenece el propietario y para todos los demás usuarios.

$755_8 = 111\ 101\ 101_2$

El grupo de 3 bits de la derecha hacen referencia al propietario, el del centro al grupo de usuarios del propietario y el de la derecha al resto de usuarios.

Cada bit puede brindar si su valor es 1 un permiso o negarlo si su valor es 0, cada grupo de 3 bits proporciona permisos de lectura, escritura y ejecución respectivamente de derecha a izquierda.

ACTIVIDAD DE APRENDIZAJE

El estudiante realizará y analizará el siguiente programa en “C” que le permite copiar un archivo a otro.

```
#include <stdio.h>
#include <fcntl.h>
#include “syscall.h”
#define PERMS 0666        /* lectura y escritura para propietario, grupo y otros */

void error(char *,…);

/* cp: copia f1 a f2 */
main(int argc, char *argv[])
        {
                int f1, f2, n;
                char buf[BUFSIZ];
                if(arg ¡= 3)
                error (“uso: cp de hacia”);
                if((f1 = open(argv[1], O_RDONLY, 0)) == -1)
                        error(“cp: no se puede abrir %s”, argv[1]);
                if((f2 = creat(argv[2],PERMS)) == -1)
                        error(“cp: no se puede crear %s, modo %03o”,
                        argv[2],PERMS);
                while((n = read(f1, buf, BUFSIZ)) > 0)
                        if (write(f2, buf, n) != n)
                                error(“cp: error de escritura en archivo %s”, argv[2]);
                return 0;
        }
```

Este programa crea el archivo de salida con permisos fijos O666, se podría utilizar el la llamada al sistema stat para determinar el modo de un archivo existente para dar el mismo modo a la copia.

El estudiante investigará cómo funciona la llamada al sistema stat.

7.5.3 Close

Existe un límite en el número de archivos que un programa en “C” puede tener abiertos simultáneamente, comúnmente de 20. Por esta razón es muy importante la siguiente función, cuya sintaxis es:

```
close(int fd)
```

Esta función suspende la conexión entre un descriptor de archivo y un archivo abierto, liberando al descriptor de archivo para que pueda ser utilizado por otro archivo. Es muy similar a la función fclose de la biblioteca estándar, pero no existe un buffer que vaciar. La función exit o return desde el programa principal cierra todos los archivos abiertos,

7.5.4 Unlink

La sintaxis de esta función es:

```
unlink(char *nombre)
```

remueve el archivo nombre del sistema de archivos.

Corresponde a la función remove de la biblioteca estándar.

7.5.5 Acceso aleatorio: Lseek

Normalmente tanto la entrada como la salida son secuenciales, pero “C” también soporta archivos de E/S de acceso aleatorio mediante el sistema de E/S/ de bajo nivel mediante llamadas a lseek(), su formato general es el siguiente:

```
int fd, origen;
long desplazamiento;
lseek(fd, desplazamiento, origen);
```

En donde fd es un descriptor de archivo devuelto al llamar a creat() o a open().

La forma en que trabaja depende de los valores del origen y del desplazamiento. El origen puede ser 0, 1 o 2, en la siguiente tabla se especifica cómo se interpreta el desplazamiento para cada uno de estos 3 valores:

Origen	Efecto al llamar a lseek()
0	Cuenta el desplazamiento desde el principio del archivo.
1	Cuenta el desplazamiento desde la posición actual del archivo.
2	Cuenta el desplazamiento desde el final del archivo.

ACTIVIDAD DE APRENDIZAJE

El estudiante realizará un programa para leer archivos, la finalidad de dicho programa debe ser definida por el profesor.

AUTOEVALUACIÓN

1. Menciona 3 funciones útiles en el manejo de archivos.

   open, close, creat, unlink.

2. ¿Cuáles son los dispositivos estándar para la entrada y salida?

   El teclada para la entrada y la pantalla para la salida.

3. ¿Qué es un descriptor de archivo?

   Siempre que vamos a realizar operaciones de entrada / salida sobre un archivo se utiliza el “descriptor de archivo” como identificación en lugar del nombre de archivo, un descriptor de archivo es similar al apuntador de archivo usado por la biblioteca estándar.

4. Escribir una llamada a scanf() para que introdusca una cadena de caracteres y dos números enteros.

   scanf(“%s%d%d, s, d1, d2);

5. Escribir una llamada a printf() que muestre en pantalla por lo menos 5 caracteres, pero no mas de 10, para cada uno de los siguientes tipos de datos: entero, cadena de caracteres, y flotante.

   printf(“%5.10d %5.10s %5.10f”, dec, string, flt);

BIBLIOGRAFÍA

- El Lenguaje de Programación C, Brian W. Kernighan, Dennis M. Ritchie, Prentice-Hall Hispanoamericana, S.A., Segunda Edición, México 1988
- Lenguaje C Programación Avanzada, Herbert Schildt, Mc Graw Hill, México, 1990
- Metodología de la Programación, Luis Joyanes Aguilar, Mc Graw Hill, México,1987
- Programación en Lenguaje C, Herbert Schildt, Mc Graw Hill, México, 1990

GLOSARIO

Acceso Aleatorio: en computación es la habilidad que se tiene para acceder a un elemento arbitrario de una secuencia de datos, no requiere haber accesado al elemento previo.

Apuntadores: variable existente que hace referencia a una dirección física de memoria de una computadora

Archivo: es un conjunto de bits almacenado en un dispositivo periférico de una computadora.

Arreglos: tabla de una o más dimensiones que permite almacenar de manera ordenada información en una computadora.

Bit: acrónimo de Binary Digit, es un dígito del sistema de numeración binario, solo puede tener dos valores 0 o 1, es la unidad básica de información en una computadora.

Byte: Secuencia de bits contiguos, en muchas ocasiones conjunto de 8 bits, es la unidad básica de almacenamiento en una computadora.

Compilar: Traducir un programa escrito en un lenguaje de programación generalmente de alto nivel a código de máquina que puede ser interpretado por una computadora.

Constante: es un valor que no puede ser alterado durante la ejecución de un programa.

Función: es un subprograma que sirve para resolver una tarea específica, en programación son subrutinas que devuelven un valor.

Ligar: tomar los programas en código objeto o de máquina generado por un compilador para unirlo con todos los códigos de funciones, subrutinas, etc. almacenados en las librerías para generar el código ejecutable de un programa.

Operador: es un símbolo matemático que indica que debe ser llevada a cabo una operación específica sobre un cierto número de operandos.

Recursividad: es la forma en la cual se especifica un proceso basado en su propia definición, en programación es un proceso que dentro de su definición hace llamada a el mismo.

Variable: es un símbolo que representa un elemento no especificado de un conjunto dado, en programación es una zona de memoria que puede tomar diversos valores durante la ejecución de un programa.