UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

SIMULADOR DE COMPUERTAS LÓGICAS USANDO LENGUAJE C

T E S I S
QUE PARA OBTENER EL TITULO DE
INGENIERO MECÁNICO ELÉCTRICO
AREA: ELÉCTRICA- ELECTRÓNICA

P R E S E N T A:

ARMANDO NAVA LINARES

2006

INDICE

INTRODUCCIÓN

Introducción

Este trabajo propone la creación de un simulador de compuertas lógicas, para resolver diagramas lógicos simples. ¿Para qué crear un simulador si ya existen varios otros?, fundamentalmente por el proceso que se tiene que seguir, mismo que no es revelado por los simuladores comerciales, en los cuales no es posible integrar nuevas funciones o características por nosotros mismos, ya que no se da a conocer el código fuente. El conocimiento del código fuente, abre las puertas a la construcción de muchas más funciones para integrarlas a nuestros programas, tanto de nuestra parte como por parte de otros programadores.

Para crear el simulador se utilizó un lenguaje de programación de carácter general, como lo es el lenguaje C, la versión elegida posee funciones para desplegar gráficos en la pantalla de la computadora, lo cual es fundamental ya que se pretende que el usuario tenga participación con el simulador. La característica de esta versión del lenguaje C para desplegar gráficos es poco conocida, por lo que se expondrá brevemente como aprovecharla y hacer uso de ella.

La simulación por computadora proporciona muchas ventajas, por ejemplo, no son necesarios elementos físicos para poder recrear algún fenómeno, podemos repetir la simulación muchas veces, se pueden alterar parámetros y ver como influyen en nuestra simulación, etc. Otra característica de la simulación por computadora es que causa un gran impacto visual en el usuario, manteniéndolo interesado en la solución del problema.

La computadora digital es uno de los grandes inventos de la actualidad, su aparición ha llevado a grandes avances en casi todos los campos, la computadora no solo permite la rápida manipulación de información, sino que también nos ayuda a interpretarla. Los bloques lógicos con los que se construye y diseña están compuestos por compuertas lógicas.

Las compuertas lógicas son la base de cualquier sistema digital, la gran mayoría de las veces no están presentes en forma aislada, sino que forman estructuras lógicas más complejas. El conocimiento de cómo funcionan es parte fundamental, en el estudio de la electrónica digital, a partir de estas bases, se pueden desarrollar sistemas más y más complejos.

CAPITULO 1

COMPUERTAS LÓGICAS

1.1 INTRODUCCIÓN

En 1854 George Boole introdujo un tratamiento sistemático de la lógica y desarrollo para este propósito un sistema algebraico que ahora se conoce como álgebra booleana. En 1938 C.E. Shannon introdujo un álgebra booleana de dos valores denominada álgebra de interruptores, en la cual demostró que las propiedades de los circuitos eléctricos y estables con interruptores pueden representarse con esta álgebra.

Las funciones Booleanas pueden implementarse de manera práctica usando compuertas electrónicas, también conocidas como, compuertas lógicas mismas que están presentes en sistemas que requieren hardware digital electrónico, desde una calculadora electrónica, hasta el más complejo sistema digital, siendo el más representativo la computadora digital, la cual ha hecho posible muchos avances en casi todos los campos.

Se puede decir que las compuertas lógicas son los ladrillos de todo sistema electrónico digital, ya que a partir de las compuertas lógicas básicas, se construyen otros elementos más complejos como flip-flops, sumadores, restadores, registros, memoria RAM, etc.

1.2 COMPUERTAS LÓGICAS.

Una compuerta lógica lleva a cabo una determinada función lógica f(x), en base a sus valores de entrada, los cuales pueden tener solo uno de dos estados (0/1, ALTO/BAJO, FALSO/VERDADERO).

Podemos ver a una compuerta como una caja negra, de la cual sabemos con anterioridad que resultado tendremos dada una entrada, y desconociendo el como se llega a ella.

Las compuertas lógicas básicas son AND, OR y NOT. Estas compuertas son usadas en diferentes combinaciones, las cuales permiten a una computadora ejecutar operaciones usando lenguaje binario.

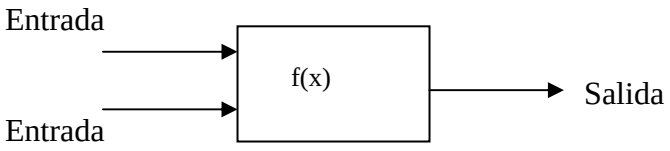


Fig. 1.1 Compuerta

Una compuerta lógica tiene por lo menos una entrada (usualmente más de una) y exactamente una salida. Físicamente se puede representar una compuerta mediante interruptores, para una compuerta AND, se tiene el siguiente circuito:

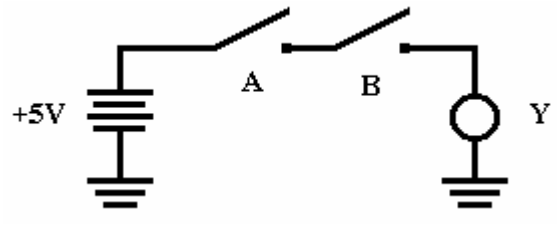


Fig. 1.2 Implementación de una compuerta AND

Analizando el circuito, la lampara solo encenderá si los interruptores A y B se encuentran cerrados.

Un ejemplo práctico de este circuito, es el interruptor principal de nuestras casas al que llamaremos A y el interruptor de nuestra sala B, por ejemplo, tendremos luz en nuestra sala si y solo si ambos interruptores están cerrados. Los interruptores A y B podrían estar en dos posiciones posible ABIERTO ó CERRADO, las combinaciones posibles son:

Interruptor A	**Interruptor B**	**Lámpara Y**
ABIERTO	ABIERTO	APAGADO
ABIERTO	CERRADO	APAGADO
CERRADO	ABIERTO	APAGADO
CERRADO	CERRADO	ENCENDIDO

Tabla 1.1 Estado de los interruptores y salida de una compuerta AND

Usando valores binarios tenemos:

Entrada A	Entrada B	Salida Y
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 1.2 Entradas y salida de la compuerta AND

La tabla 1.2 es la tabla de verdad de la compuerta AND, la cual define las posibles salidas en base a las entradas A y B. Su símbolo es:

Fig. 1.3 Símbolo de la compuerta AND

Para una compuerta OR nuestro circuito queda de la siguiente manera:

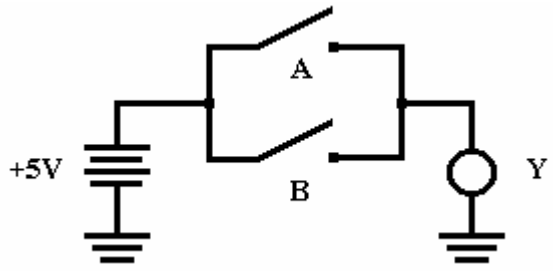


Fig. 1.4 Implementación de una compuerta OR mediante interruptores

En nuestro circuito la lámpara encenderá, si alguno ó ambos interruptores se encuentran cerrados, el único caso en el que la lámpara estará apagada es cuando ambos interruptores se encuentran abiertos.

Una posible analogía de este circuito en mundo real, serian nuestros ojos, cuando los mantenemos abiertos percibimos imágenes (entrada de luz), podemos cerrar alguno de ellos (suspendemos el estímulo) y seguir viendo imágenes, solo cuando mantenemos cerrados ambos dejamos de ver.

Las combinaciones posibles de los interruptores A y B en la figura 1.3 son:

Interruptor A	**Interruptor B**	**Lámpara Y**
ABIERTO	ABIERTO	APAGADO
ABIERTO	CERRADO	ENCENDIDO
CERRADO	ABIERTO	ENCENDIDO
CERRADO	CERRADO	ENCENDIDO

Tabla 1.3 Estado de los interruptores y salida de una compuerta OR

Usando valores binarios tenemos:

Entrada A	**Entrada B**	**Salida Y**
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 1.4 Entradas y salida de la compuerta OR

Su símbolo es:

Fig. 1.5 Símbolo de la compuerta OR

Para una compuerta NOT tenemos el siguiente circuito:

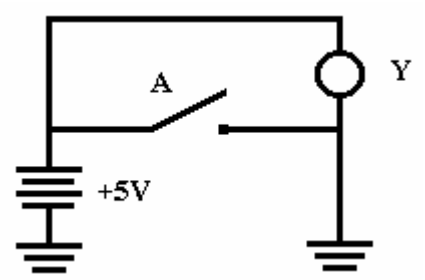


Fig. 1.6 Implementación de un inversor

Las dos compuertas descritas anteriormente poseen cada una dos entradas y una salida. La compuerta NOT o inversora, posee una entrada y una salida. Su función es producir una salida inversa o contraria a su entrada es decir convertir unos a ceros y ceros a unos. Nuestra analogía para esta compuerta es por ejemplo, una balanza, si de una lado agregamos peso, el otro sube, mientras que si quitamos peso, el otro baja.

Interruptor A	**Lámpara Y**
ABIERTO	ENCENDIDO
CERRADO	APAGADO

Tabla 1.5 Estado del interruptor y salida del inversor

Usando valores binarios tenemos:

Entrada A	**Salida Y**
0	1
1	0

Tabla 1.6 Entradas y salida del inversor

Tiene el siguiente símbolo:

Fig. 1.7 Símbolo del inversor

Hasta este momento se han mencionado 3 tipos de compuertas, sin embargo existen otras más. Limitaremos nuestro alcance a compuertas con *n* entradas.

Las tablas de verdad para cada compuerta tiene 2^n líneas. Una compuerta esta definida completamente por la columna de salida en la tabla de verdad, esta columna puede ser vista como una cadena de 2^n dígitos binarios. Ya que hay 2^k diferentes cadenas de *k* dígitos binarios , y si $k=2^n$, entonces hay 2^{2^n} cadenas de salida.

Utilizando todas las posibles combinaciones de dos variable binarias (n=2), se tienen en total 16 combinaciones ó funciones lógicas posibles ($F_0 - F_{15}$).

x	**y**	**F_0**	**F_1**	**F_2**	**F_3**	**F_4**	**F_5**	**F_6**	**F_7**	**F_8**	**F_9**	**F_{10}**	**F_{11}**	**F_{12}**	**F_{13}**	**F_{14}**	**F_{15}**
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Tabla 1.7 Funciones para dos variable binarias x, y.

Cada una de estas funciones define a un tipo diferente de compuerta, muchas no tienen ningún nombre, ya que carecen de uso. Analizando cada cadena de salida o función tenemos lo siguiente:

- 0000 Una compuerta que ignora sus entradas y siempre proporciona un 0 en la salida. Esta compuerta no requiere ningún circuito, solo hay que dejas conectada la salida a 0.
- 0001 Esta es la compuerta AND descrita anteriormente.
- 0010 Esta es como una compuerta AND con un inversor en la segunda entrada (y).
- 0011 Esta compuerta ignora la segunda entrada (y), y tiene como valor de salida, el valor de la primera entrada(x). La salida va conectada a la primer entrada (x).
- 0100 Esta es como una compuerta AND con un inversor en la primer entrada (x).

- 0101 Esta compuerta ignora la primer entrada (x), y tiene como valor de salida, el valor de la primera segunda(y). La salida se conecta a la segunda entrada (y).
- 0110 Se conoce como compuerta OR-EXCLUSIVA (cualquiera pero no ambos).
- 0111 Es una compuerta OR descrita anteriormente.
- 1000 Es la combinación de una compuerta OR con un inversor a su salida (NOR).
- 1001 Es una compuerta OR-EXCLUSIVA con un inversor a su salida.
- 1010 Esta compuerta pude ser construida con in inversor en la segunda entrada (y).
- 1011 Es una compuerta OR con un inversor en la segunda entrada (y).
- 1100 Esta compuerta pude ser construida con in inversor en la primer entrada (x).
- 1101 Es una compuerta OR con un inversor en la primer entrada (x).
- 1110 Es una compuerta AND con un inversor a su salida (NAND).
- 1111 Esta compuerta siempre proporciona un 1 a la salida. La salida va conectada a 1.

De las 16 funciones definidas en la tabla 1.7, solo 8 se utilizan como compuertas estándar básicas. Sus nombres son: complemento (NOT), transferencia (BUFFER), AND, OR, NAND, NOR, excluyente-OR (EXOR), y equivalencia (EXNOR).

1.3 TABLAS DE VERDAD

Una tabla de verdad muestra el valor que puede tomar cada una de las variables de entrada y su relación con alguna operación lógica.

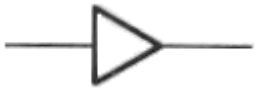
Nombre	Símbolo gráfico	Función algebraica	Tabla de verdad
AND		F=xy	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		F=x+y	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inversor		F=x’	x | F 0 | 1 1 | 0
Buffer		F=x	x | F 0 | 0 1 | 1
NAND		F= (xy)’	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		F= (x+y)’	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Excluyente-OR (XOR)		F = xy’+x’y = x ⊕ y	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Excluyente-NOR o Equivalente		F= xy + x’y’ = x ⊙ y	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Tabla 1.8 Tablas de verdad de las compuertas básicas

1.4 LÓGICA POSITIVA Y NEGATIVA

Los valores de entrada para nuestras compuertas han estado referidos como 1 y 0, físicamente debemos proporcionar un voltaje que represente dichos valores. Hay diferentes familias lógicas de compuertas, y cada posee distintas características, entre las cuales están los niveles de tensión requeridos. Tomaremos como ejemplo la familia lógica TTL.

Los voltajes con que trabaja una compuerta TTL son 0V y 5V. Los 5V son el estado alto o uno lógico, mientras que 0V nos representan el estado bajo o cero lógico.

En la lógica positiva al 1 lógico le corresponde el nivel más alto de tensión y al 0 lógico el nivel mas bajo. En la lógica negativa ocurre todo lo contrario, es decir, se representa al estado "1" con los niveles más bajos de tensión y al "0" con los niveles más altos. En el caso de otras familias lógicas los voltajes pueden ser negativos, para distinguir cuando se usa lógica positiva y lógica negativa nos debemos referir a la amplitud relativa de las señales definidas como ALTO o BAJO, (High/Low).

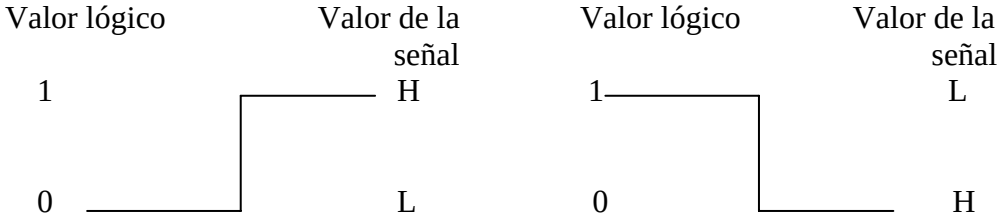


Fig. 1.8 Lógica positiva y negativa

En la practica los niveles de tensión de las entradas y salidas no son fijos, pueden existir fluctuaciones para los valores de voltaje que definen un ALTO o un BAJO. Esta consideración nos proporcionan los limites de voltaje que nos aseguran la correcta operación de nuestras compuertas. Es decir existe una tolerancia para nuestros voltajes, y una zona prohibida, en la cual no se puede garantizar un nivel lógico 1 o 0.

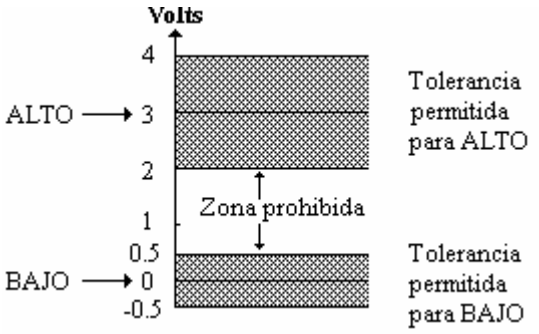


Fig. 1.9 Márgenes de voltaje y zona prohibida

En la figura 1.9 se muestra un sistema en el cual 3V representan un ALTO, mientras que 0V representan un BAJO, existe una tolerancia aceptada para estos voltajes, la zona prohibida se cruza solo durante la transición ó cambio de estado de ALTO a BAJO o viceversa. La siguiente, es una tabla comparativa de los valores de voltaje, de las familias lógicas TTL y CMOS.

Tecnología	**Zona prohibida entrada**	**Zona prohibida salida**	**V_{cc}**	**V_{IH}**	**V_{IL}**	**V_{OH}**	**V_{OL}**
TTL	0.8 a 2V	0.4 a 2.4V	5V	2 a 5.5V	0 a 0.8V	2.5 a 5.5V	0 a 0.4V
CMOS	1.5 a 3.5V	0.01 a 4.99V	3 a 15V	3.5 a 5V	0 a 1.5V	4.99 a 5V	0 a 0.01V

Tabla 1.9 Comparación entre las familias TTL y CMOS

Donde:

V_{cc} = Voltaje de alimentación de las compuertas, en CMOS se ha supuesto en 5V

V_{IH} = Voltaje de entrada para ALTO

V_{IL} = Voltaje de entrada para BAJO

V_{OH} = Voltaje a la salida para ALTO

V_{OL} = Voltaje a la salida para BAJO

V_{cc} es nuestro voltaje de alimentación, para TTL se toma el valor típico de 5 Volts. En el caso de CMOS, esta familia opera con valores de voltaje cuyo rango esta entre 3 a 15 volts, siendo valores típicos 5 y 10 volts.

Manteniendo V_{IH} en el valor mínimo de voltaje, se garantiza habrá un ALTO en la entrada de la compuerta y si V_{IL} no excede los valores máximos, se garantiza será tomado como un BAJO por la compuerta.

V_{OH} y V_{OL} son valores de voltaje ALTO y BAJO respectivamente, que nos garantiza el fabricante a la salida de la compuerta, son obtenidos bajo condiciones de prueba del fabricante.

1.5 FAMILIAS LÓGICAS

Los circuitos digitales emplean componentes encapsulados, conocidos como circuitos integrados (IC), los cuales pueden albergar puertas lógicas o circuitos lógicos

más complejos. Según el número de compuertas que contienen, tienen diferentes escalas de integración:

- SSI (Small Scale Intergration). Unas cuantas compuertas en un solo paquete
- MSI (Medium Scale Integration). De 10 a 100 compuertas.
- LSI (Large Scale Intergration). Más de 100 compuertas
- VLSI (Very Large Scale Integration). Miles de compuertas.

Las familias lógicas deben su nombre generalmente a los elementos con los que se construyen.

RTL.	Lógica Resistor-Transistor
DTL.	Lógica Diodo-Transistor
TTL.	Lógica Transistor-Transistor.
ECL.	Lógica de Emisor Acoplado.
MOS.	Semiconductor de Óxido Metálico.
CMOS.	Semiconductor Complementario de Óxido Metálico.
I^2L.	Lógica de Inyección Integrada.

Cada familia lógica tienen una compuerta básica, a partir de la cual se construyen todas las demás compuertas. Las familias RTL y DTL, fueron las primeras en implementarse y rara vez se usan en nuevos diseños. La familia TTL es una modificación de la DTL, es una de las más populares, cuenta con gran variedad de circuitos. ECL ofrece una alta velocidad, sin embargo la disipación de potencia y la inmunidad al ruido son las peores de las familias lógicas disponibles. Las familias MOS y I^2L se utilizan en circuitos que requieren alta densidad de componentes. La CMOS se emplea en sistemas de bajo consumo de potencia.

Las compuertas lógicas se pueden construir a partir de elementos tales diodos, transistores y resistencias, al utilizar estos elementos físicos lo primero en lo que hay que pensar es una fuente de alimentación y por consiguiente un consumo de potencia ó ***disipación de potencia*** por parte de dichos elemento.

También se presenta un ***retardo de tiempo***, es decir cuando se presenta una transición del valor lógico 1 al valor 0 ó viceversa, en la entrada de la compuerta, la respuesta del circuito no se ve reflejada de inmediato a la salida.

Nuestras compuertas presentan otra característica llamada, ***abanico de salida***, esto es el número de cargas estándar que la compuerta puede impulsar, sin afectar de su operación, dicho de otro modo, las compuertas solo pueden proporcionar una cantidad limitada de corriente, si se sobrepasa, se pone en riesgo su correcto funcionamiento.

Existe algo más que debemos considerar en cuanto a la operación de las compuertas, es el ***margen de ruido*** de estas con capaces de tolerar, por ruido entenderemos cualquier señal indeseable sobre la señal normal de operación.

Familia lógica IC	**Abanico de salida**	**Disipación de potencia (mW)**	**Retardo de propagación (ns)**	**Margen de ruido (V)**
DTL	8	12	30	1
Estándar TTL	10	10	10	0.4
ECL	25	25	2	0.2
CMOS	50	0.1	25	3

Tabla 1.10 Características típicas de las familias lógicas

CAPITULO 2

EL LENGUAJE DE PROGRAMACIÓN C

2.1 EL LENGUAJE C

C es un lenguaje de programación de propósito general, es decir, que no esta enfocado a una aplicación especifica, sino que puede ser utilizado para construir cualquier aplicación. La base del C proviene de los lenguajes BCPL, escrito por Martin Richards, y del B escrito por Ken Thompson en 1970 para el primer sistema UNIX en la DEC PDP-7. Estos son lenguajes sin tipos, al contrario que el C que proporciona varios tipos de datos.

C fue diseñado para el sistema operativo UNIX, por Dennis Ritchie, quien lo implanto en la DEC PDP-11. El sistema operativo, el compilador de C y esencialmente todos los programas de aplicación de UNIX están escritos en C.

Podemos decir que el lenguaje C es de relativo “bajo nivel”, ó de nivel medio, ya que combina elementos de alto nivel con la funcionalidad del lenguaje ensamblador. Esta característica no lo hace menos, al contrario, el termino “bajo”, debe ser interpretado como “profundo”, en muchos casos es más efectivo y conveniente que otros lenguajes supuestamente más poderosos.

Este lenguaje no esta ligado a ningún sistema operativo, ni a ninguna máquina en particular, por lo cual otra característica es su portabilidad, ya que permite utilizar el mismo código en diferentes equipos y sistemas informáticos: El lenguaje es independiente de la arquitectura de cualquier máquina en particular.

Originariamente, el manual de referencia del lenguaje para el público fue el libro de Kernighan y Ritchie, escrito en 1977. Es un libro que explica y justifica totalmente el desarrollo de aplicaciones en C, aunque en él se utilizaban construcciones, en la definición de funciones, que podían provocar confusión y errores de programación que no eran detectados por el compilador. Como los tiempos cambian y las necesidades también, en 1983 ANSI establece el comité X3J11 para que desarrolle una definición moderna y comprensible del C. El estándar está basado en el manual de referencia original de 1972 y se desarrolla con el mismo espíritu de sus creadores originales. La primera versión de estándar se publicó en 1988 y actualmente todos los compiladores utilizan la nueva definición. Una aportación muy importante de ANSI consiste en la definición de un conjunto de librerías que acompañan al compilador y de las funciones contenidas en ellas. Muchas de las operaciones comunes con el sistema operativo se realizan a través de estas funciones. Una colección de ficheros de encabezamiento, headers, en los que se definen los tipos de datos y funciones incluidas en cada librería. Los programas que utilizan estas bibliotecas para interactuar con el sistema operativo obtendrán un comportamiento equivalente en otro sistema.

2.2 ESTRUCTURA DE UN PROGRAMA EN C

Por experiencia propia, puedo decir que la mejor manera de aprender el lenguaje C es mediante la práctica, capturar el código, compilarlo, depurarlo y ejecutar el programa, no pueden ser sustituidos por la teoría, así que iniciaremos con un programa de ejemplo, el siguiente ejemplo muestra la estructura básica de un programa en C:

```
#include <stdio.h>                          Archivo de cabecera
main()                                      Función Principal
{                                           Inicio de la función
principal
      printf(“Hola mundo”);                 Función printf();
      // Imprime Hola mundo                 Comentario estilo C++
      /* Imprime Hola mundo */              Comentario estilo C
return 0;                                   Código de termino del
programa

}                                           Fin de la función principal
```

Ahora explicaremos cada uno de los elementos:

Archivo de cabecera	Contiene las definiciones para funciones especificas, en este caso printf();
Función principal	La función principal main() es un elemento que debe estar presente en todo programa escrito en C, ya que indica el principio y fin del programa. Nos permite además llamar a otras funciones.
Inicio de la función principal	Se indica con un corchete {
Función printf()	Es una función que nos permite desplegar una cadena de texto en la pantalla.
Comentario	Este elemento es un auxiliar, su uso no es obligatorio, pero nos es útil para indicar quien es el autor, que hace el programa, la fecha de creación, etc. Puede ser estilo C ó estilo C++
Código de termino del programa	Es un valor de retorno, puede ser utilizado para indicar el estatus de termino del programa.
Fin de la función principal	Se indica con un corchete }

Tabla 2.1 Partes básicas de una programa en C

La estructura general de un programa en C queda de la siguiente manera

```
declaraciones globales
main{
variables locales
bloque
}

función1()
{
Variables locales
bloque
}
```

Las declaraciones globales son validas durante toda la vida de nuestro programa, mientras que las variables locales solo son reconocidas por la función en donde se declaran, más allá no existen, los bloques son las instrucciones de nuestro programa, en C podemos crear nuestras propias funciones y mandarlas llamar cada vez que sean necesarias.

2.3 VARIABLES Y TIPOS DE DATOS.

Las variables son elementos del lenguaje C que nos permiten almacenar datos de diferentes tipos, el tipo de dato varia de compilador en compilador, es decir no todos los lenguajes maneja los mismos tipos de datos.

2.3.1 Variables

Una variable es una localidad de memoria, que contiene un determinado valor, el tipo de variable indica como se manejara la información contenida en la variable. Para dar nombre a una variable se utiliza un identificador, un identificador es una secuencia de caracteres, que nos sirve manejar a una variable

REGLAS BÁSICAS PARA NOMBRAR IDENTIFICADORES
1. Secuencia de letras o dígitos; el primer carácter puede ser una letra o un subrayado
2. Los identificadores son sensibles a las mayúsculas `minum` es distinto de `MiNum`
3. Los identificadores pueden ser de cualquier longitud, pero solo son significativos los 32 primeros (compiladores bajos DOS: Borland y Microsoft)
4. Los identificadores no pueden ser palabras reservadas

Tabla 2.2 Reglas para nombrar identificadores

2.3.2 Palabras reservadas

Los siguientes identificadores no pueden usarse para nombrar variables, debido que son utilizadas por el compilador para llevar a cabo diferentes tareas.

asm	default	friend	protected	switch	void
auto	delete	goto	public	template	volatile
break	do	if	register	this	while
case	double	inline	return	throw	
catch	else	int	short	try	
char	enum	long	signed	typedef	
class	extern	new	sizeof	union	
const	float	operator	static	unsigned	
continue	for	private	struct	virtual	

2.3.3 Tipos de datos

En C existen básicamente 4 tipos de variables o tipos de datos.

TIPO	TAMAÑO	RANGO DE VALORES
Char	1 byte	-128 a 127
Int	2 bytes	-32,768 a +32,767
Float	4 bytes	$3.4x10^{-38}$ a $3.4x10^{+38}$
bouble	8 bytes	$1.7x10^{-308}$ a $3.4x10^{+4932}$

Tabla 2.3 Tipos de datos básicos

Tipo char o carácter

Este tipo puede contener un único carácter de código ASCII, es decir un carácter un dígito numérico o un signo de puntuación, en c un carácter es tratado en todo como un número, su uso es apto para almacenar números pequeños.

Tipo int o entero

Las variables int contienen números enteros (sin decimales), dentro del limite de su tamaño, este depende de la plataforma del compilador y del numero de bits que se use por palabra de memoria 8, 16,32.....

Tipo float o coma flotante

Las variables de este tipo almacenan números en formato de coma flotante, mantisa y exponente. Las variables de este tipo almacenan números con decimales, son aptos para números grandes, pero el fuerte de estas variables no es la precisión, sino el orden de magnitud, es decir lo grande o pequeño que es el número que contiene. Los formatos en coma flotante sacrifican precisión en favor de tamaño

Tipo double o coma flotante con doble precisión

Las variables de este tipo almacenan números en formato de coma flotante, mantisa y exponentes, , al igual que float, pero usan mayor precisión. Son aptos para variables de tipo real. Usaremos estas variables cuando trabajemos con números grandes, pero también necesitemos gran precisión

2.3.4 Calificadores de tipo

Sirven para modificar el rango de valores de un determinado tipo de variable

signed. Le indica a la variable que va a llevar signo. Es el utilizado por defecto.

signed char	1 byte	-128 a 127
signed int	2 bytes	-32,768 a +32,767

unsigned . Le indica a la variable que no va a llevar signo. Valor absoluto

unsigned char	1 byte	0 a 256
unsigned int	2 bytes	0 a 65535

short. Rango de valores en formato corto (limitado). Es el utilizado por defecto

short char	1 byte	-128 a 127
short int	2 bytes	-32, 768 a +32767

long. Rango de valores en formato largo (ampliado)

long int	4 bytes	-2147483648 a 2147483637
long double	10 bytes	$3.4x10^{-4932}$ a $1.1x10^{+4932}$

También es posible combinar calificadores entre si

signed long int = long int = long		
unsignes long int = unsigned log	4 bytes	0 a 4,294,967,295

2.3.5 Declaración de variables

Antes de poder usar una variable se debe declarar, es decir enunciar el tipo y el nombre de la variable.

```
int i;   //Declara una variable entera de nombre i
```

También es posible declarar múltiples variables en una sola instrucción, como se muestra en la siguiente declaración.

```
float uno, dos, tres;   //Declara las variables
                        //uno, dos y tres
                        //como tipo float
```

Según el lugar donde se declaren las variables pueden ser de dos tipos, globales ó locales.

Una variable global se declara antes de main(), puede ser usada durante todo el programa y se destruye al termino del mismo.

Una variable local se declara después de la función main, adentro de una función donde vaya a ser utilizada y se destruye al termino de la misma

2.3.6 Inicialización de variables.

Una variable puede ser inicializada, es decir se le puede asignar un valor, en el momento en que se declara o durante la ejecución del programa.

`int i;`
`float numero;`
 	
`i=1;`
`numero=23.56;`
	`int i=1;`
	`float numero= 23.56;`

Tabla 2.4 Inicialización de variables

2.3.7 Constantes

Al contrario de las variables las constantes no pueden ser alteradas durante la ejecución del programa, para indicarle al compilador que se trata de una constante usaremos la directiva #define, ejemplo:

```
#define PI 3.14159
#define NOMBRE PEDRO
```

Esta directiva no solo sustituye un identificador por un valor numérico, sino también por una cadena.

<Directiva> <Identificador> <Valor>

2.4 OPERADORES

Un operador nos permite llevar a cabo una operación matemática o lógica en el lenguaje C, todo esto mediante una ecuación generalmente, en la que tenemos *operandos* y un *operador.*

<Operando1> <Operador> <Operando2> 4+3 ejemplo de operador binario

<Operando1> <Operador> -x ejemplo de operador unario

2.4.1 Operador de asignación

El operador = asigna el valor de la expresión derecha a la variable situada a su izquierda, este operador es asociativo por la derecha, eso permite realizar asignaciones múltiples.

```
grados = 25.48;
```

La expresión

```
a = b = c = 35;
```

equivale a

```
a = (b = (c = 35) );
```

El operador de asignación =, en C++ proporciona en combinación con otros, cinco operadores de asignación adiciones.

Símbolo	Uso	Equivale a	Descripción
=	a = b;	a = b;	Asigna el valor de b a a
*=	a *= b;	a = a * b;	Multiplica a por b y asigna el resultado a la variable a
/=	a /= b;	a = a / b	Divide a por b y asigna el resultado en la variable a
%=	a %= b;	a = a % b;	Fija a al resto de a/b
+=	a += b;	a = a + b;	Suma b y a y lo asigna a la variable a
-=	a -= b;	a = a – b;	Resta b de a y asigna el resultado a la variable a

Tabla 2.5 Operadores de asignación

El uso de estos operadores depende en gran medida de estilo de escritura que el programador este acostumbrado.

2.4.2 Operadores Aritméticos

Este tipo de operadores nos sirve para realizar operaciones aritméticas básicas

Operador	Tipos enteros	Tipos reales	Ejemplo
+	Suma	Suma	4 + 5
-	Resta	Resta	7 – 3
*	Producto	Producto	4 * 5
/	División entera: cociente	División en coma flotante	8 / 5
%	División entera: resto		9 % 5

Tabla 2.6 Operadores aritméticos

Los operadores aritméticos C++ siguen las reglas típicas de jerarquía o prioridad. Estas reglas especifican la precedencia de las operaciones aritméticas. Ejemplo:

En la expresión: 3 + 5 * 2 , ¿El resultado será?

1) 8*2 = 16

2) 3+10 = 13

De acuerdo con las reglas de precedencia, la multiplicación se realiza antes que la suma, la expresión original 3 + 5 * 2 equivale a:

3 + (5 * 2)

En C++ las expresiones interiores a paréntesis se evalúan primero; a continuación se realizan los operadores unarios, seguidos por los operadores de multiplicación, división, resto, suma y resta.

Operador	**Operación**	**Nivel de precedencia**
+, -	+25, -6.543	1
*, /, %	5 * 6 es 30	2
+, -	2 + 3 es 5 2 - 3 es 1	3

Tabla 2.7 Precedencia de operadores matemáticos básicos

Cuando los operadores + y – se utilizan delante de un operador actúan como operadores unarios mas y menos

```
+25              //significa que es positivo
-6.543           //significa que es negativo
```

Ejemplo. ¿Cuál será el resultado de la expresión: 6 + 2 * 3 – 4 /2 ?

6 + **2 * 3** – 4 /2

6 + 6 – **4 /2**

6 + 6 – 2

12 – 2

10

2.4.3 Operadores de Asociatividad

La asociatividad nos permite alterar el orden en que se efectúan las operaciones mediante los paréntesis, en la siguiente expresión

3 * 4 + 5

Se efectúa primero la multiplicación ya que tiene prioridad sobre la suma, de tal manera que el resultado es 17.

Si queremos efectuar primero la suma y después la multiplicación, debemos hacer lo siguiente:

3 * (4 + 5)

El resultado será entonces 27.

La asociatividad determina el orden en que se efectuaran las operaciones cuando se tienen operadores de igual prioridad, ya se a derecha a izquierda, o de izquierda a derecha.

10 – 5 + 3 se agrupa como (10 – 5) + 3

ya que la suma y la resta tienen igual prioridad, pero tienen asociatividad de izquierda a derecha. Sin embargo,

x = y = z

se agrupa como

x = (y = z)

ya que su asociatividad es de derecha a izquierda

Prioridad (mayor a menor)	Asociatividad
+, - (unarios)	Izquierda-derecha (→)
*, /, %	Izquierda-derecha (→)
+, -	Izquierda-derecha (→)

Tabla 2.8 Prioridad y Asociatividad

2.4.4 Operadores de incremento y decremento

Los operadores ++ y --, suman o restan 1 cada vez que se aplican a una variable

Incremento	Decremento
++n	-- n
n += 1	n -= 1
n = n + 1	n = n+1

Tabla 2.9 Operadores incremento (++) y decremento (--)

De tal manera que

a++

Es igual que

a +1

Las sentencias

++n; //incrementa n antes de que su valor se utilice

n++; //incrementa n después de que su valor se ha empleado.

tienen el mismo efecto, incrementar n; así como

--n;

n--;

Sin embargo en un contexto donde el valor esta siendo usado y no solo el efecto ++n y n++ los resultados son diferentes. Si los operadores ++ y -- están de prefijos, la operación de incremento se efectúa antes que la de asignación; si los operadores ++ y -- están de sufijos, la asignación se efectúa en primer lugar y el incremento o decremento a continuación.

```
int a=1, n;
n = ++a;    // n vale 2 y a vale 2
```

```
int a=1, n;
n = a++;    // n vale 1 y a vale 2
```

2.4.5 Operadores relacionales

Los operadores relacionales se usan para comparar una condición, tal como igualdad, desigualdad y diferencias relativas. Son utilizados normalmente con las sentencias (if, while, for). Cuando se utilizan operadores en una expresión, el operador relacional produce un 0 ó un 1, dependiendo del resultado de la condición. El valor 0 es resultado de una condición falsa, mientras que 1 se devuelve cuando la condición es verdadera. En el siguiente ejemplo la variable *a* toma el valor de 1 ya que 3 es menor que 7 y la variable *b* toma el valor de 0 dado que 2 no es mayor que 5.

```
a = 3 < 7;
b = 2 > 5;
```

Operador	**Significado**	**Ejemplo**
==	Igual a	a==b
!=	No igual a	a != b
>	Mayor que	A > b
<	Menor que	A < b
>=	Mayor o igual que	A >= b
<=	Menor o igual que	A <= b

Tabla 2.10 Operadores relacionales en C++

Los operadores relacionales tienen menor prioridad que los operadores aritméticos y asociatividad de izquierda a derecha.

m + 5 <= 2 * n equivale a (m + 5) <= (2 * n)

Los operadores relacionales permiten comparar dos valores, por ejemplo:

if (valor_tot < 10) Comprueba si la variable valor_tot es menor que 10

if (valor_tot == 10) Comprueba si la variable valor_tot es igual que 10

if (valor_tot != 10) Comprueba si la variable valor_tot es diferente de 10

2.4.6 Operadores lógicos

Estos operadores se devuelven un valor verdadero (cualquier valor entero distinto de cero) o un valor falso (0). Los operadores lógicos se denominan también *operadores booleanos,* en honor de George Boole, creador del álgebra de Boole.

Los operadores en C++ son: **not (!), and (&&), y or (||)**. El operador lógico **not** produce falso si su operado es verdadero (distinto de cero) y viceversa. **and** produce verdadero si ambos operandos son verdaderos. **or** produce verdadero si cualquiera de sus operandos es verdadero y produce falso, solo si, ambos operandos son falsos.

Operador lógico	**Operación lógica**	**Ejemplo**
Negación (!)	No lógica	! (x >= y)
Y lógica (&&)	Operando_1 && operando_2	m<n && i>j
O lógica (||)	Operando_1 || operando_2	m=5 || n!=10

Tabla 2.11 Operadores lógicos

De la misma forma que con los operadores aritméticos, el resultado de una expresión formada con operadores lógicos depende del operador y de los operandos. En el caso de los operadores lógicos solo son posibles dos valores como resultado verdadero (cualquier valor distinto de cero) y falso (0). La forma más usual de representar el resultado de las operaciones lógicas es mediante tablas de verdad.

Operando (a)	**NOT a**
Verdadero (1)	Falso (0)
Falso (0)	Verdadero (1)

Tabla 2.12 Tabla de verdad para el operador logico NOT

Operandos		**Operador AND**
A	B	A&&b
Verdadero (1)	Verdadero (1)	Verdadero (1)
Verdadero (1)	Falso (0)	Falso (0)
Falso (0)	Verdadero (1)	Falso (0)
Falso (0)	Falso (0)	Falso (0)

Tabla 2.13 Tabla de verdad para el operador lógico AND

Operandos		**Operador OR**
A	B	A||b
Verdadero (1)	Verdadero (1)	Verdadero (1)
Verdadero (1)	Falso (0)	Verdadero (1)
Falso (0)	Verdadero (1)	Verdadero (1)
Falso (0)	Falso (0)	Falso (0)

Tabla 2.14 Tabla de verdad para el operador lógico OR

Los operadores relacionales se utilizan en expresiones condicionales y mediante las sentencias if, while o for. Por ejemplo

```
if ( (a<b) && (c>d) )
{
cout << “Los resultados no son validos”;
}
```

Si la variable a es menor que b y al mismo tiempo c es mayor que d, entonces se visualiza el mensaje: Los resultados no son validos.

El operador ! tienen prioridad más alta que &&, que a su vez tiene mayor prioridad que ||. La asociatividad es de izquierda a derecha.

La precedencia de los operadores es la siguiente: los operadores aritméticos tienen precedencia sobre los operadores relacionales, y los operadores relacionales tienen precedencia sobre los operadores lógicos.

2.5 ENTRADA / SALIDA

Al igual que en el C ANSI, las funciones de entrada/salida no están definidas ene el propio lenguaje C++, sino que están incorporadas bajo la forma de bibliotecas. En C existe la biblioteca stdio.h estandarizada por el ANSI; en C++ su biblioteca correspondiente es iostream.h. Se puede recurrir a cualquiera de estas bibliotecas sin embargo iostream.h tiene ventajas sobre stdio.h, ya que la entrada /salida se realiza por flujos (streams), que ofrece gestión de clases, sobrecarga de operadores y funciones. Una

ventaja más es que se pueden manipular las operaciones E/S sin necesidad de conocer los conceptos típicos de orientación de objetos.

2.5.1 Flujos estándar

Un flujo es una corriente de datos, entre una fuente y un destino. Entre y origen y destino debe de existir una conexión o tubería (“pipe”) por la que circulan estos datos. Estas conexiones se realizan mediante operadores (<< y >>) y funciones E/S.

La biblioteca iostream define cuatro flujos estándar:

- El flujo cin, definido por la clase istream, está conectado al periférico de entrada estándar (el teclado, representado por el archivo STDIN).
- El flujo cout, definido por la clase ostream, esta conectado al periférico de salido estándar (la pantalla, representado por el archivo STDOUT).
- El flujo cerr, definido por la clase ostream, esta conectado al periférico de error estándar (la pantalla, representado por el archivo STDOUT). Este flujo no es a través del buffer.
- El flujo clog, definido por la clase ostream, esta conectado igualmente al periférico de error estándar (la pantalla, representado por el archivo STDOUT). Al contrario que cerr, el flujo clog se realiza a través de buffer.

cerr tiene ventaja sobre clog debido a que cada vez que se utiliza el flujo de salida se limpia y la salida esta disponible más rápidamente en el dispositivo externo. Sin embargo en grandes cantidades de mensajes, la versión clog es más eficiente.

C++ visualiza las entradas y salidas como flujos de caracteres. Cuando se tienen entradas desde el teclado, un archivo de disco, un módem o un ratón, C++ ve solo un flujo de caracteres pero no reconoce cual es el dispositivo que le proporciona la entrada.

Estas operaciones E/S de flujos utilizan las mismas funciones, tanto como para obtener la entrada de teclado como la de un módem. Se pueden utilizar las mismas funciones para escribir en un archivo de disco, una impresora o una pantalla. Pero es necesario un medio para encaminar el flujo de entrada o salida al medio adecuado.

2.5.2 Salida a la pantalla

Para enviar información de salida a la pantalla se utiliza el objeto cout. Mediante el operador de inserción <<. El operador << dirige el contenido de la variable a su derecha al objeto (cout) de su izquierda.

```
cout << “Hola mundo, C++”
```

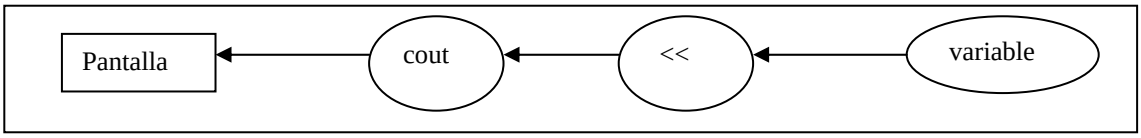


Tabla 2.15 Salida con cout

El operador de inserción se define para todos los tipos de datos básicos. El operador de inserción convierte los datos a la derecha de << al tipo de dato esperado por el objeto cout.

```
int salida = 32;

cout << salida;
```

El valor entero 32 contenido en la variable salida, se convierte en la cadena 32 y se pasa al objeto cout.

2.5.3 Operadores de inserción en cascada.

El operador de inserción de puede poner en cascada, de modo que podemos colocar muchos elementos en una sentencia C++, la sentencia.

```
cout << 1 <<  2 << 3 << 4 << 5;
```

Genera la salida

1 2 3 4 5

Cuando se trata de enviar caracteres a la pantalla, estos deben estar entre comillas, además los operadores en cascada pueden mezclar valores de caracteres y numéricos, por ejemplo:

```
int  numero = 4;
cout << “El valor es: ” << numero << endl;
```

Se visualizara la siguiente salida

El valor es: 4

La instrucción endl hace que el flujo avance a la siguiente línea.

2.5.4 Lectura del teclado.

La sentencia en C++ que se utiliza para la lectura de datos es cin, que al igual que cout, es un objeto predefinido en C++ y es parte de la biblioteca oistream.h.

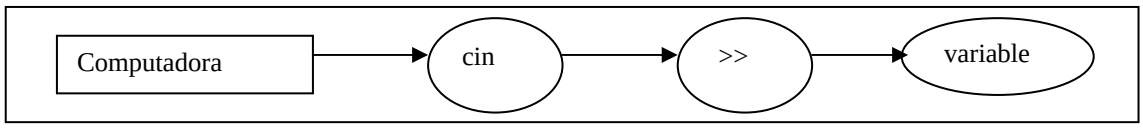


Tabla 2.16 Entrada con cin

El operador extracción >> se usa para manejar la entrada de un flujo. El operador obtiene los datos del flujo y lo sitúa en una variable. El operador extracción se define también para todos los tipos de datos básicos. Al igual que el operador de inserción, el operador de extracción se puede poner en cascada.

```
int edad;
float altura;
cout << “Introduzca Edad y altura”;
cin >>Edad >> Altura;
```

2.5.5 Lectura de datos carácter.

Los caracteres se leen uno a uno de acuerdo con las reglas siguientes:

1.- Los blancos (espacios en blanco, tabulaciones, nuevas líneas y avances de pagina) son ignorados por cin cuando se utiliza el operador >>.

2.- Los valores numéricos se pueden leer como caracteres, pero cada dígito es un carácter independiente.

```
#include<iostream>
void main()
{
      char letra1;
      char letra2;
      char letra3;
cin >>letra1>> letra2 >>letra3;
cout <<letra1 <<letra2 <<letra3;

}
```

Este programa lee tres caracteres y los almacena el las variables letra1, letra2 y letra3, si se siguen tecleando más datos el programa los ignora, únicamente toma los tres primeros.

2.5.6 Lectura de datos cadena.

Cuando deseamos leer un dato de tipo cadena (más de una letra), debemos asignarle espacio a este dato, por ejemplo:

```
#include<iostream>
void main()
{
char nombre[30];  //Reservamos un espacio de 30 caracteres
                  // en la variable nombre
cout << “Escriba es su nombre: ”
cin >> nombre >>endl;
}
```

Si la cadena de entrada contiene más de una palabra, el objeto cout no leerá más que la primera palabra, debido a que se encontrará con un espacio en blanco. Si la cadena de entrada excede el tamaño reservado, solo se guardaran las letras correspondientes al espacio reservado.

2.5.7 Bases de numeración.

Normalmente los enteros se visualizan como números decimales (base 10), se pueden visualizar los datos en otra base de numeración distinta (octal -8-, hexadecimal -16-) mediante las funciones dec(), hex(), o bien oct(). Por ejemplo:

```
cout << oct << Total << endl; //El valor de Total se visualiza en octal
cout << hex << Total << endl; //El valor de Total se visualiza en
hexadecimal
```

Existe el manipulador setbase(int n) que establece la base numérica 8,10 o 16. Funciona igual que los manipuladores 8, 10 o 16, por ejemplo.

```
cout << setbase(10);
```

2.6 CONTROL DE FLUJO

Las instrucciones para el control de flujo especifican el orden en que se realizaran los procesos. Una expresión como x=0 ó i++ se convierte en una instrucción cuando va seguida de punto y coma, por ejemplo:

```
x = 0 ;
i++;
```

En C, el punto y coma es un terminador de la instrucción, junto con las instrucciones de control de flujo se usan las llaves {}, que delimitan un bloque o instrucciones compuestas, no se requiere punto y coma después de la llave derecha que termina un bloque.

2.6.1 La instrucción if-else

Esta instrucción se utiliza para tomar decisiones, su sintaxis es

if (expresión)

Instrucción1

else

Instrucción2

La instrucción if evalúa la expresión, si esta es verdadera (diferente de cero)se ejecuta la Instrucción1, de lo contrario se ejecuta la Instruccion2 que esta después de else. La parte else de un if-else es optativa, existe una ambigüedad cuando un else se omite en una secuencia if anidada.

```
if (n > 0)
      if(a>b)
            Z=a;
      else
            Z=b;
```

El else va asociado con el if más interno. Si esto no es lo que se desea, se deben utilizar llaves para forzar la asociación correcta.

```
if (n > 0)  {
      if(a>b)
            Z=a;
}
else
      Z=b;
```

En este caso mediante el uso de las llaves, el else va asociado al primer if

2.6.2 Else if

En la construcción

if (expresión)

Instrucción

else if(expresión)

Instrucción

else if(expresión)

Instrucción

else if(expresión)

Instrucción

else

Instrucción

Las expresiones se evalúan en orden, de ser verdaderas, la instrucción asociada con ella se ejecuta, el código de cada instrucción puede ser una instrucción simple o un grupo de instrucciones dentro de llaves. Esta construcción es la forma más general de escribir una decisión múltiple. El ultimo else maneja el caso “ninguno de los anteriores” o caso por omisión, cuando ninguna de las expresiones se satisface.

2.6.3 switch

La instrucción switch es una toma de decisión múltiple, que prueba si una expresión coincide con un número de valores constantes enteros y traslada el control adecuadamente.

```
switch(expresión){
        case exp-const: Instrucciones
        case exp-const: Instrucciones
        default: instrucciones
}
```

Cada case se etiqueta con un valor contante entero o expresión contante entera. Si un case coincide con el valor de la expresión, la ejecución comienza ahí. Todas la expresiones case deben ser diferentes. La etiqueta default se ejecuta si ningún case se satisface, el default es optativo, en caso de no encontrarse y si ningún case se satisface, no se toma acción alguna.

Los case solo sirven como etiquetas, si se satisface la condición, se ejecuta el código y luego la ejecución pasa al siguiente, a menos que se tome una acción especifica para terminar el switch. Las formas más comunes de dejar un switch son **break** y **return**.

Esta característica de pasas a través de los case permite conectar varios case en una acción simple, pero también implica que cada case debe terminar con un break. El siguiente ejemplo ilustra lo anterior

Usando break	**Sin usar break**

`#include<stdio.h>`
`#include<stdlib.h>`
`#include<conio.h>`
 	
`void main(){`
 	
`char vocal;`
`int es_vocal;`
`printf("Teclee una letra: \n");`
`scanf("%c", &vocal);`
 	
`switch(vocal){`
`case 'a':`
`case 'e':`
`case 'i':`
`case 'o':`
`case 'u':`
`es_vocal=1;`
`break;`
`default:`
`es_vocal=0;`
`}`
`printf("Valor: %d \n", es_vocal);`
 	
`getch();`
`}`
	`#include<stdio.h>`
	`#include<stdlib.h>`
	`#include<conio.h>`
	
	`void main(){`
	
	`char vocal;`
	`int es_vocal;`
	`printf("Teclee una letra: \n");`
	`scanf("%c", &vocal);`
	
	`switch(vocal){`
	`case 'a':`
	`case 'e':`
	`case 'i':`
	`case 'o':`
	`case 'u':`
	`es_vocal=1;`
	`// break;`
	`default:`
	`es_vocal=0;`
	`}`
	`printf("Valor: %d \n", es_vocal);`
	
	`getch();`
	`}`

Tabla 2.17 Uso de break

El ejemplo del lado izquierdo emplea break para terminar el switch, si se teclea una vocal, la variable es_vocal toma el valor de 1. En el ejemplo del lado derecho la instrucción break esta comentada, de tal forma que es_vocal primero toma el valor de 1 ya que se satisface la condición, pero como el flujo continua, después toma el valor de 0.

2.7 BUCLES

Los bucles o ciclos son el núcleo de cualquier lenguaje de programación y se encuentran presentes en la mayor parte de ellos, nos permiten llevar a cabo tareas repetitivas.

2.7.1 Ciclo while

Es la sentencia de bucle más sencilla

while (condición) Instrucción

La instrucción se ejecuta mientras la condición sea verdadera, si la primera vez que se evalúa la condición, esta resulta falsa, la instrucción nunca se ejecuta. Por ejemplo:

```
while (x < 100) x = x + 1;
```

Mientras la variable x sea menor que 100, su valor se incrementara en 1.

2.7.2 Ciclo do–while

Su sintaxis es la siguiente:

do

Instrucción

while(expresión)

A diferencia de while, do-while ejecutara por lo menos una vez la instrucción, ya que la expresión a ser evaluada por while esta al final, el ciclo se repetirá mientras la expresión sea verdadera. Ejemplo:

```
do
x = x + 1;
while (x < 100);
```

x se incrementara en 1 hasta que valga 100.

2.7.3 Ciclo for

La sintaxis es:

for(expresión1;expresión2; expresion3)

Instrucción

Donde *expresión1* corresponde a una inicialización o declaración de la variable a usar, si se declaran en este punto solo tendrán validez dentro del bucle for. La instrucción se ejecutara repetitivamente, hasta que la evaluación de la *expresión2* resulte ser falsa. Por último la *expresión3* corresponde a un incremento o decremento de la variable a usar. Ejemplo:

```
for(int i = 0; i < 10; i = i + 1)              cout << “Hola \n”;
```

Se inicializa primero la variable 1 a la vez que se declara, mientras i sea menor que 10, el valor de i se incrementara en 1, y se imprimirá en la pantalla la palabra Hola. Todas las expresiones son opcionales, se puede simular un ciclo while, o hacer bucles infinitos

```
for(;i < 100;) i = i +1;      //simulación del bucle while
for( ; ; )      //Bucle infinito
```

2.7.4 Break y continue

La sentencia break se utiliza cuando se requiere salir de un ciclo, sin probar al inicio o fin de este, proporciona una salida anticipada a un for, while y do, tal como lo hace en el switch. Ejemplo:

```
y = 0;
x = 0;
while(x < 1000)
{
        if(y == 1000) break;
        y++;
}
x = 1;
```

El bucle no terminaría nunca de no ser por la sentencia break, ya que la variable x nunca cambia de valor. El uso de la sentencia continue, provoca que inicie la siguiente iteración del ciclo for, while o do, que la contiene. Por ejemplo:

```
for (i=0; i< 100 ; i++){
      if(i<90)
            continue;   //ignora los elementos menores a 90
      printf( “El valor de i es:  %i \n”, i);
}
```

En este ejemplo se ignoran los elementos menores que 90, tras lo cual se imprimen valores del 90 en adelante, hasta en 99.

2.7.5 La sentencia de salto goto

El uso de goto no es aconsejable en la practica, su uso más común es abandonar el procesamiento de una estructura profundamente anidada, tal como salir de dos o más ciclos a la vez. La instrucción break no puede ser usada directamente, puesto que solo sale del ciclo más interno.

```
for (...){
        for(...){
                ...
                if(desastre)
                goto error;
}
...
error:
```

Como puede apreciarse goto hace referencia a una etiqueta (error:), que tiene la misma forma que un nombre de variable y es seguida por dos puntos.

2.8 FUNCIONES

Para entender el concepto de una función, debemos recordar que ya trabajamos con una función, su nombre es main, como señalamos, esta es la función principal en un programa en C, sabemos entonces que una función contiene sentencias ó instrucciones, declaraciones, comentarios, etc. C++ es un lenguaje de programación modular, lo cual permite dividir un programa grande, en módulos (rutinas pequeñas de denominadas funciones), esto trae como ventajas aislar los posible errores, ya que solo se tiene que buscar el problema en una parte del código, y por consecuencia, el programa en más fácil de mantener.

2.8.1 Estructura de una función

Una función, es un conjunto de instrucciones que se pueden llamar desde cualquier parte del programa. Las funciones no se pueden declarar dentro de otra función. En C++ todas las funciones son externas o globales. La estructura de una función es la siguiente:

```
TipoDeValorDeRetorno    NombreDeLaFunción(ListaDeParametros)
{
Cuerpo de la función
Valor de retorno
}
```

- El tipo de valor de retorno, nos indica el tipo de dato que la función devuelve.
- El nombre de la función es un identificador, con el cual podemos llamar a la función.
- La lista de parámetros, es una serie de variables tipificadas (con tipos), muy similar a la declaración de una variable.
- El cuerpo de la función contiene las instrucciones a ejecutar por la función.
- El valor de retorno, es un valor devuelto por la función.

2.8.2 Nombre de una función.

Los nombres de las funciones empiezan con una letra o subrayado (_) puede contener tantas letras, números o subrayados como se desee, sin embargo en el caso de Turbo C++, el compilador ignora a partir de 32. C++ es sensible a las mayúsculas, esto significa que las letras mayúsculas y minúsculas son distintas cuando se emplean para dar nombre a una función. Construyamos una función para sumar dos números.

```
float suma (float a, float b)
{
float add;
      add = a+b;
return add;
}
```

Como podemos observar, la lista de parámetros consta de las variables a y b, del tipo float, mismas que son utilizadas por nuestra función, esto se denomina paso de

parámetros, dentro de nuestra función se declara una variable *add*, que se usa para almacenar nuestro resultado, una vez efectuadas las instrucciones de la función lo ultimo que hace la función es devolver un valor, en este caso el valor contenido en la variable *add*.

2.8.3 Tipo de dato de retorno

El tipo de dato de retorno de nuestra función es del tipo float, ya que este tipo de dato no permite sumar cualquier numero, ya sea entero o coma flotante. Cuando no se especifica el tipo de valor de retorno de una función, el compilador asume que se trata del tipo int. Cuando una función no devuelve ningún valor, se puede utilizar el tipo void para indicarle al compilador que la función no devuelve ningún valor. Por ejemplo:

```
void  VerResultados(int a, int b);
```

2.8.4 Resultados de una función

La función suma, proporciona un valor de retorno, que es el valor contenido en la variable add de tipo float. Una función devuelve un único valor, mediante la sentencia return, cuya sintaxis es.

return(expresión)

Ejemplo:

```
return(a+b+c);
```

El valor devuelto(expresión) puede ser cualquier tipo de dato, excepto una función o un array.

Una función puede tener cualquier número de sentencias return, tan pronto como la función encuentra una sentencia return, devuelve el valor indicado por esta. Si el tipo de valor de retorno es void, return puede escribirse como:

```
return;
```

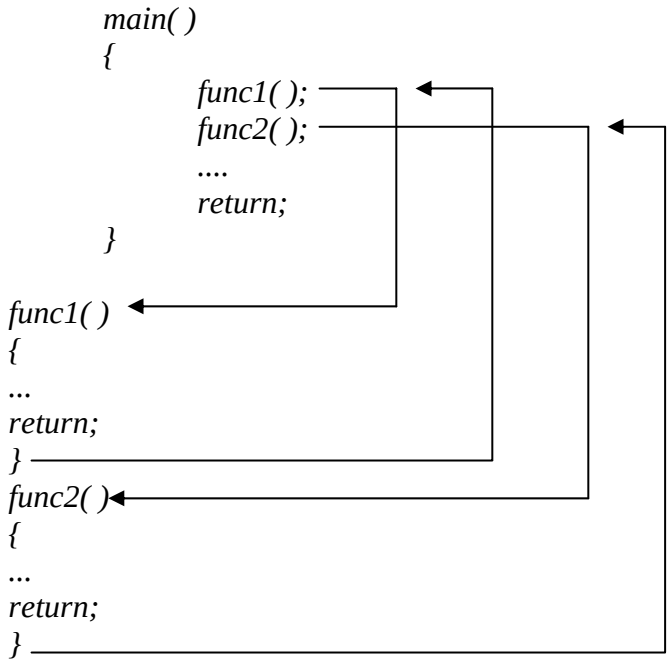
El valor de vuelto suele encerrase entre paréntesis, pero su uso es opcional. Cuando de antemano sabemos que nuestra función no devuelve ningún valor, podemos omitir la sentencia return.

```
void funcion1(void)
{
      cout<< “No se devuelven valores”;
}
```

2.8.5 Llamada a una función.

Para usar una función, esta debe ser llamada o invocada, cualquier expresión puede contener una llamada a una función, cuando se llama a una función, el flujo del programa se dirige a la función llamada.

Las funciones pueden ser llamadas desde la función main pero también desde otras funciones.



2.8.6 Prototipos de las funciones

C++ requiere que una función se declare o se defina antes de su uso, esta declaración recibe el nombre de prototipo de la función. Un prototipo consta de los siguientes elementos: tipo de valor de retorno nombre de la función, una lista de argumentos encerrados entre paréntesis y un punto y coma

```
float MiFuncion(float a, float b);
```

Los prototipos se sitúan normalmente al principio de un programa, antes de la definición de cualquier función. El compilador utiliza los prototipos para verificar el numero y los tipos de datos con los argumentos formales en la función llamada.

Un ejemplo de prototipo es el siguiente:

```
int proceso(int a, char b, float c, double d, char e);
```

El compilador utiliza solo la información de los tipos de datos, por lo cual los nombres de los parámetros no son necesarios y pueden ser omitidos. El ejemplo anterior puede escribirse

```
int proceso(int , char, float, double, char);
```

El objeto de escribir el nombre de los prototipos, es hacer la declaración más fácil de leer y escribir.

Vamos a retomar nuestra función suma en el siguiente ejemplo.

```
#include<iostream.h>
#include<conio.h>
float SUMA(float a, float b); //Prototipo de suma
main()
{
float oper1, oper2,resultado;
clrscr();                          //Borramos la pantalla
cout << "Teclee dos operandos a sumar"<< endl;
      cout<< "Operando 1: ";
      cin >> oper1;
            cout<<"Operando 2: ";
            cin >> oper2;
      resultado = SUMA(oper1, oper2); //Llamamos a SUMA
      cout << "La suma de: ";
      cout << oper1 <<" mas "<< oper2 <<" Es: "<< resultado;
getch();
return 0;
}

float SUMA (float a, float b)
{
```

```
float add;
            add = a+b;
return add;
}
```

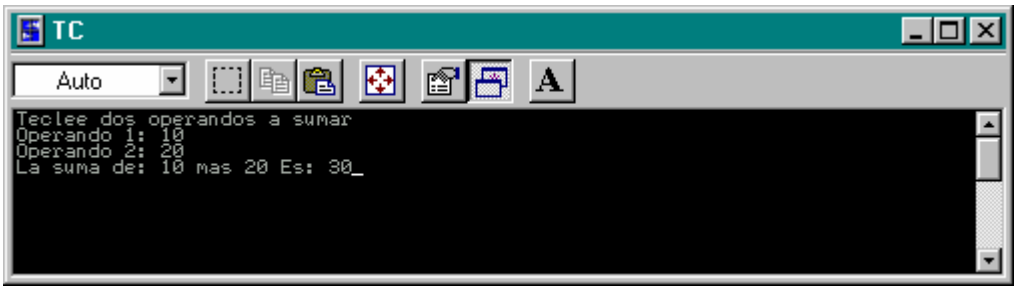


Fig. 2.1 Salida de la función suma

Aquí tenemos la salida de nuestro programa. En el programa principal solicitamos dos operandos a sumar, los cuales usamos como parámetros de nuestra función SUMA, la cual los procesa y nos devuelve al valor de la variable add, mismo que asignamos a la variable resultado, por ultimo el programa principal visualiza el resultado en pantalla.

2.8.7 Paso de parámetros a una función

C++ nos proporciona dos formas de pasar parámetros a una función, la primer forma ya la usamos en la función SUMA y fue mediante *parámetros por valor* , la otra forma es mediante parámetros por referencia.

2.8.8 Paso de parámetros por valor.

Se llama también *paso por copia*, esto significa que el compilador hace una copia de los valores de los parámetros y los pasa a la función. Si se cambia el valor de la variable local, el efecto no se percibe fuera de la función.

En esta técnica de paso de parámetro por valor, la función receptora no puede modificar la variable que recibe.

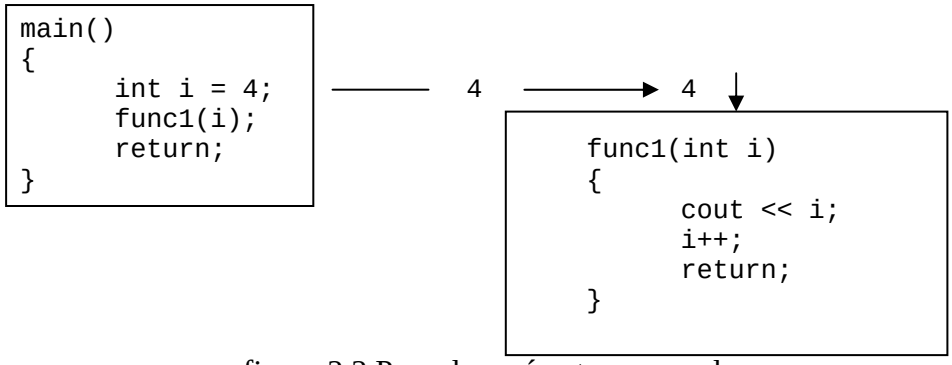


figura 2.2 Paso de parámetros por valor

2.8.9 Paso de parámetros por referencia.

En este método es posible modificar el valor del parámetro pasado y devolver ese valor modificado a la función llamadora. También se le conoce como paso de parámetros por dirección.

El compilador pasa la dirección del parámetro a la función llamadora, cuando se modifica el valor de la variable, este valor queda almacenado en la misma dirección. Para declarar a una variable del tipo parámetro por referencia, el símbolo & debe preceder a la variable.

Por cuestiones de compatibilidad, C++ conserva el uso de punteros para implementar parámetros por referencia.

```
/* Método de paso por referencia estilo C */
void Intercambio(int* a, int* b)
{
      int aux = *a;
           *a = *b;
           *b = aux;
}
```

Implementando la función Intercambio tenemos lo siguiente:

```
//Programa Intercambio
#include<iostream.h>
void Intercambio(int*, int*);

main()
{
```

```
  int i=33, j=44;
       cout << endl;
       cout << "El valor de i es: "<<i << " El valor de j es: "<<
j<<endl;

       Intercambio(&i, &j);

       cout << "El valor de i es: "<<i << " El valor de j es: "<<
j<<endl;

return 0;
}

void Intercambio(int* a, int* b)
{
       int aux = *a;
            *a = *b;
            *b = aux;
}
```

La función Intercambio() pasa las direcciones de i y j a los punteros de tipo entero a y b, después se intercambian los valores contenidos en i y j, el cambio tiene efecto sobre estas variables ya que los que se pasa a la función Intercambio() es la dirección en donde se encuentran las variables y no solo su valor.

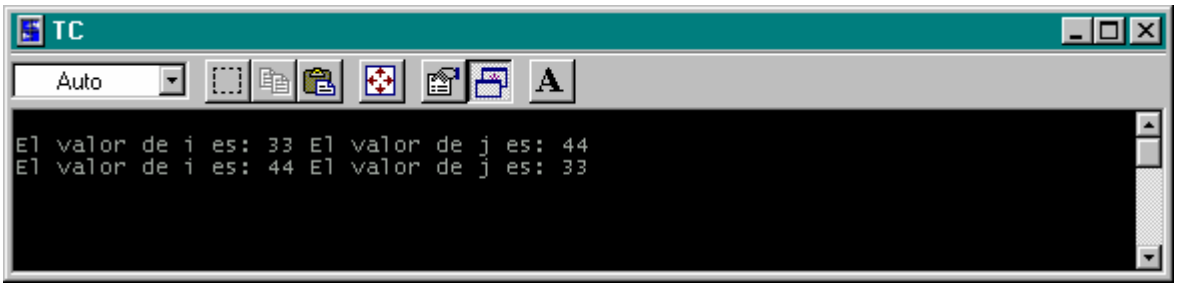


Figura 2.3 Ejemplo de la función Intercambio().

La siguiente versión de la función Intercambio() que utiliza parámetros por referencia es la siguiente.

```
void Intercambio2(int& a, int& b)
{
      int aux = a;
            a = b;
            b = aux;
}
```

En la función Intercambio2(), los parámetros a y b son parámetros referenciados por lo cual cualquier cambio que tenga dentro de la función, se transmitirá al exterior de la misma.

```
//Programa Intercambio2
#include<iostream.h>
```

```
void Intercambio(int*, int*);
void Intercambio2(int&, int&);

main()
{
     int i=33, j=44;
     cout << endl;
cout << "El valor de i es: "<<i << " El valor de j es: "<< j<<endl;

      Intercambio(&i, &j);
cout << "El valor de i es: "<<i << " El valor de j es: "<< j<<endl;
      cout <<endl;

      Intercambio2(i, j);
cout << "El valor de i es: "<<i << " El valor de j es: "<< j<<endl;
            cin.get();
return 0;
}

void Intercambio(int* a, int* b)
{
      int aux = *a;
           *a = *b;
           *b = aux;
}

void Intercambio2(int& a, int& b)
```

```
{
      int aux = a;
            a = b;
            b = aux;
}
```

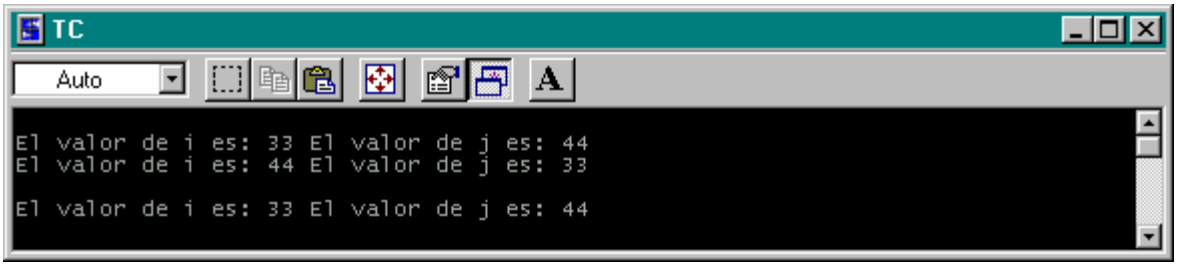


Figura 2.4 Salida del programa intercambio2

Mediante la función Intercambio(), se intercambian los valores de i por j, posteriormente la función Intercambio2() los restaura a su valor original.

2.9 ARRAYS

Una array o arreglo esta formado por un conjunto de datos del mismo tipo, para acceder a cada elemento se usan subíndices. Normalmente se utiliza para almacenas tripos tales como int, char o float.

2.9.1 Declaración de un array.

De la misma forma que con las variables, los arrays deben ser declarados antes de utilizarlos. Su sintaxis es la siguiente

Tipo Identificador [num_elem][num_elem][num_elem]..... ;

Donde:

Tipo. Indica el tipo de dato de nuestro array

Identificador. Es el nombre de nuestro array

Num_elem. Nos indica el numero de elementos de nuestro array.

Los valores para el número de elementos deben ser constantes, se pueden declarar arrays de múltiples dimensiones. Cuando se usan de una sola dimensión se suele hablar de listas o vectores, cuando se usan de dos, se trata de tablas.

```
int lista[10];
```

Los elementos de lista serán:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Como se observa. Los subíndices deben ser del tipo entero, y toman valores desde 0 hasta (numero de elementos)-1. Hecho que deberá ser tomado muy en cuenta. Cada elemento del array puede contener su propio valor.

Si queremos acceder a un elemento del array únicamente tenemos que usar el subíndice adecuado. Por ejemplo para el primer y ultimo elemento del array lista []:

```
int PRIMERO=10, ULTIMO=20;
lista[0]= PRIMERO;
lista[9]=ULTIMO;
```

C++ no comprueba que los subíndices del array estén dentro del valor que le definimos, por lo cual deberemos tomar la precaución de no usar elementos no definidos en nuestro array, por ejemplo lista[], solo contempla diez elementos definidos por los subíndices 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9, el uso de la expresión lista[15] no provocaría error en compilador, pero puede producir un fallo en el programa.

2.9.2 Arrays multidimensionales

Los arrays de una sola dimensión se caracterizan por tener un solo subíndice. Los multidimensionales son aquellos que tienen más de una dimensión y por consecuencia más de un subíndice para referirse a sus elementos. No es usual requerir arrays de más de dos o tres dimensiones.

Para un array de 2 dimensiones de tendrá:

```
float matriz[4][3];
```

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]
[3][0]	[3][1]	[3][2]

La posición de los elementos en el array obedece a la regla

[renglón][columna]

2.9.3 Inicialización de arrays

Los arrays pueden ser inicializados en su declaración. Ejemplos

```
char nombre[]="Laura";
char abc[]={'A','B','C'};
float data[]={34, 24,11, 3, 7, 10};
int N[5]={1,2,3,4,5};
int M[][3]={7, 3, 45, 2, 9, 35, 1, 8, 23};
```

En los tres primeros casos no es obligatorio especificar el tamaño para la primera dimensión. Cuando la dimensión queda indefinida, se calcula a partir del numero de elementos en la lista de valores iniciales.

Además del operador de asignación utilizado para asignar valores a un array, hay un operador muy útil que podemos utilizar con arrays, se trata de sizeof(). El operador sizeof(); devuelve el tamaño en bytes de la expresión o dato que se use como parámetro. En el caso de una array, podemos saber el numero de elementos, dividiendo el tamaño total del array, entre el tamaño de uno de los elementos.

```
int array[10];
      cout << "El numero de elementos es: " << endl;
      cout << sizeof(array)/sizeof(array[0])<< endl;
```

2.10 ESTRUCTURAS

Una estructura permite agrupar varios elementos, aunque sean de distinto tipo, se usa este tipo de agrupación cuando los elementos tienen algún tipo de relación. Las estructuras son llamadas también registros. Y son estructuras análogas en muchos aspectos a los registros de las bases de datos, siguiendo esta analogía, cada variable de una estructura se denomina a menudo campo.

Su sintaxis es la siguiente:

struct <Identificador de la estructura>

{

<tipo de dato miembro> <nombre miembro_1>

<tipo de dato miembro> <nombre miembro_2>

....

....

<tipo de dato miembro> <nombre miembro_n>

}

En primer lugar tenemos la palabra reservada struct, seguida del identificador o nombre de la estructura, entre llaves se encuentran los tipos de datos miembro seguidos de su identificador.

2.10.1 Definición de variables de estructuras.

Cuando declaramos una variable, lo hacemos mediante el tipo de dato y su identificador, en el caso de la definición de variables de estructuras, se crea un área en memoria reservada para los datos de la estructura declarada. Las variables de estructuras se pueden definir de dos formas: 1) listándolas a continuación del cierre de la llave de la declaración de la estructura, o 2) listando el nombre de la estructura seguida por el nombre de las variables, en cualquier lugar del programa antes de utilizarlas.

1.
```
struct CD_ROOM
{
      char titulo[30];
      char artista[30];
      int n_canciones;
      float precio;
}cd1, cd2, cd3;
```

2.
```
CD_ROOM cd1, cd2, cd3;
```

En el lenguaje C es obligatorio el uso de la palabra struct en la definición de las variables.

```
struct CD_ROOM cd1, cd2, cd3;
```

También es posible crear un array de structuras

```
struct CD_ROOM caja[200];
```

2.10.2 Inicialización de una declaración de estructuras.

La inicialización se puede hacer de dos formas. Se puede inicializar una estructura dentro de la sección de código del programa, o como parte de su declaración, en este caso la sintaxis es la siguiente:

struct <nombre variable estructura> = {

Valor miembro_1,

Valor miembro_2,

.....

Valor miembro_n};

Ejemplo:

```
struct cd_room
{
      char titulo[35];
      char artista[30];
      int año;
      char disquera[15];
}cd1 = {“Animal Nocturno”,
         “Ricardo Arjona”,
        1993,
        sony};
```

2.10.3 Acceso a estructuras.

Accedemos a una estructura para almacenar información en ella o para recuperar información. Podemos acceder a los miembros de una estructura de dos formas: 1) utilizando el operador punto (.), o 2) utilizando el operador puntero -> .

Podemos almacenar datos en las variables miembros de una estructura por medio de la inicialización, pero además podemos usar la inicialización directa y la lectura del teclado. La sintaxis mediante el operador punto en C++ es la siguiente:

<nombre variable estructura>.<nombre miembro> = datos;

Ejemplo:

```
Cd1.titulo = “All the way”;
Cd1.artista = “Celine Dion”;
Cd1.año =  1999;
Cd1.disquera = sony;
```

En el caso del operador puntero, primero de debe definir una variable puntero a la estructura, después se utiliza el operador puntero para apuntar al miembro dado de la estructura. Su sintaxis es la siguiente:

<puntero estructura> -> <nombre miembro> = dato;

Primero declaramos el puntero a la estructura

```
CD_ROOM *pcd;
```

Usamos nuestro puntero junto con el operador -> de la siguiente manera:

```
pcd -> titulo = “All the way”;
pcd -> artista = “Celine Dion”;
pcd -> año =  1999;
pcd -> disquera = sony;
```

Para usar nuestras estructuras por medio del teclado, utilizamos las sentencias de entrada y el operador punto o puntero.

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

main()
{
int DATE;
      struct CD_ROOM{
            char titulo[35];
            char artista[30];
            int year;
            char disquera[15];
            }cd1;

cout << "\n DATOS DEL CD \n";
cout << "Cual es el titulo: ";
gets(cd1.titulo);
```

```
      cout << "Nombre del artista: ";
      gets(cd1.artista);

cout << "A¤o del CD: ";
cin >> cd1.year;

      cout << "Nombre de la disquera: ";
      gets(cd1.disquera);
      cout <<endl <<endl;

DATE = cd1.year;
cout <<" LOS DATOS DEL CD SON: \n";
cout <<"TITULO: "<< cd1.titulo << endl;
cout <<"ARTISTA: "<< cd1.artista << endl;
cout <<"AÑO: " << DATE << endl;
cout <<"DISQUERA: "<< cd1.disquera << endl;
return 0;
}
```

Al final del programa podemos ver la manera de mostrar la información de una estructura, para lo cual utilizamos alguna sentencia de salida. Podemos asignar la información del miembro de una estructura a otra variable, o enviarla directamente a la salida.

1. *<Nombre variable> = <nombre variable estructura>.<nombre miembro>;*

   <Nombre variable> = <puntero de estructura>.<nombre miembro>;

2. *cout << <nombre variable estructura>.<nombre miembro>;*

   cout << <puntero de estructura>.<nombre miembro>;

Mediante el operador puntero nuestro ejemplo queda como sigue:

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

main()
{
int DATE;
      struct CD_ROOM{
            char titulo[35];
            char artista[30];
            int year;
            char disquera[15];
            };

CD_ROOM *CD;

cout << "\n DATOS DEL CD \n";
cout << "Cual es el titulo: ";
gets(CD -> titulo);
```

```
      cout << "Nombre del artista: ";
      gets(CD -> artista);

cout << "A¤o del CD: ";
cin >> CD -> year;

      cout << "Nombre de la disquera: ";
      gets(CD -> disquera);
      cout <<endl <<endl;
DATE= CD -> year;
cout <<" LOS DATOS DEL CD SON: \n";
cout <<"TITULO: "<< CD -> titulo << endl;
cout <<"ARTISTA: "<< CD -> artista << endl;
cout <<"A¥O: " << DATE << endl;
cout <<"DISQUERA: "<< CD -> disquera << endl;
return 0;
}
```

2.11 APUNTADORES

Los apuntadores son un tema que si bien es complicado, forma una parte muy poderosa del lenguaje C y C++, una buena comprensión y dominio del tema nos proporcionara una herramienta de gran potencia para desarrollar nuestros programas.

Ya hemos visto lo que son las variables, estas nos permiten guardar y manipular un valor, mediante los operadores de asignación y aritméticos.

```
int a = 20;
float b = 250.25;
char c = 'G';
```

Los punteros son un tipo especial de variable, la cual contiene la dirección en memoria de una variable, imaginemos que queremos llegar a la CNDH (Comisión Nacional de los Derechos Humanos), la forma más fácil es conociendo la "dirección" de sus oficinas Periférico Sur 3469, Colonia San Jerónimo Lidice.

Por supuesto que si no supiéramos la dirección, y decidiéramos salir en su búsqueda inmediatamente, nos seria mucho más difícil llegar.

Algo similar sucede en la computadora, para el compilador es mucho más fácil manipular nuestras variables si de antemano conoce su dirección en memoria.

Podemos ver la memoria de la computadora como un gran array en donde se encuentran datos y programas, tanto del usuario como del sistema operativo.

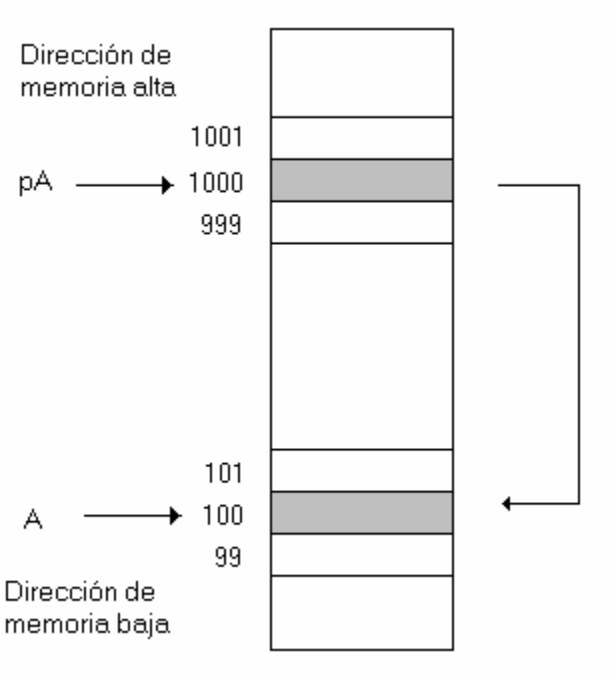


Figura 2.5 Punteros en memoria

pA es un puntero que contiene la dirección de memoria de la variable A, si la dirección de memoria de la variable A es 100 el contenido del apuntador pA será la dirección 100.

2.11.1 Declaración de punteros

Al igual que cualquier variable, los punteros deben ser declarados antes de poder usarlos. La sintaxis para la declaración de punteros es la siguiente:

*tipo *identficador;*

Obviamente el tipo corresponde al tipo de dato que queremos manejar, * es un operador de indirección , y el identificador el nombre con que identificaremos a nuestro puntero. La expresión:

```
int *p;
```

Es equivalente a

```
int* p;
```

Se lee como “p” es un puntero a un entero. Algunos ejemplos de variables punteros son:

```
int *ptr1;  //puntero a un tipo de dato entero
long *ptr2; //puntero aun tipo de dato largo (long int)
char *prt3; //puntero a un tipo de dato char
float *f    //puntero a un tipo de dato float
```

2.11.2 Punteros a Variables

Después de declarar un puntero, el siguiente paso es incializar la variable puntero con la dirección en memoria de alguna variable. Para saber la dirección de memoria de cualquier variable usamos el operador de dirección & (dirección de). Los tipos de variable tienen que ser compatibles con el tipo de puntero que usemos para interactuar

con ellas, no podemos almacenar la dirección de una variable tipo float en un puntero de tipo char.

Por ejemplo:

```
int a;            //Declaración de la variable a
int *pA;    //Declaración del puntero pA a un entero
pA = &a;    //Asignación de la dirección de memoria de la variable a en
el puntero pA
```

Para tener acceso al contenido de la variable usaremos el operador de indirección (*), por ejemplo:

```
int edad;         //Declaración de la variable edad
int *p_edad       //Declaración del puntero p_edad
p_edad =&edad;    //La dirección de memoria de edad es asignada al
                  //puntero p_edad
cout << edad;     //Imprimimos el valor de edad
cout << *p_edad   //Imprimimos el valor de edad (operador de
indirección *)
//Programa puntero.cpp
#include<stdio.h>

main(void)
{
  int A;
```

```
  int *pA;


  A = 23;
  printf("El valor de A es: %i \n", A);


  pA = &A;    //Se asigna la dirección de A al puntero pA
  *pA= *pA+1; //Incrementamos el valor de A


  printf("El valor de A es: %i \n", *pA);


  getchar();
return 0;
}
```

Figura 2.6 Salida del programa puntero

CAPITULO 3.

RECURSOS GRÁFICOS

3.1 INTRODUCCIÓN

Una de las razones por las cuales se trabajo con el compilador TC++ 3.0 de Borland, es porque esta distribución trae incorporados los recursos necesarios para desplegar gráficas en la pantalla de la computadora, además de que ofrece ejemplos para cada una de las funciones, y una gran cantidad de información que por si sola nos permite aprender a utilizar gráficas en nuestros programas o aplicaciones, faltando solo unos pequeños detalles para lograr aprovechar estos recursos.

3.2 LA BGI Y SU CONFIGURACIÓN

La BGI (Borland Graphics Interface) Interfaz Gráfica de Borland, Es una librería gráfica que se distribuye junto con le compilador TC++ 3.0 de Borland, la cual nos permite desplegar gráficas en la pantalla. Esta característica no se encuentra presente en el compilador de forma inmediata, sino que se deben de seguir una serie de pasos, debido a esto la capacidad del compilador para desplegar gráficos por lo regular se desconoce, y por lo tanto no es aprovechada, aún y cuando se cuenta con un gran conjunto de instrucciones gráficas y ayuda para poder usarlas.

Para habilitar las librerías gráficas se debe de hacer lo siguiente: en el menú Options, señalar Linker, después Libraries.

habilitar [x] Graphics Library

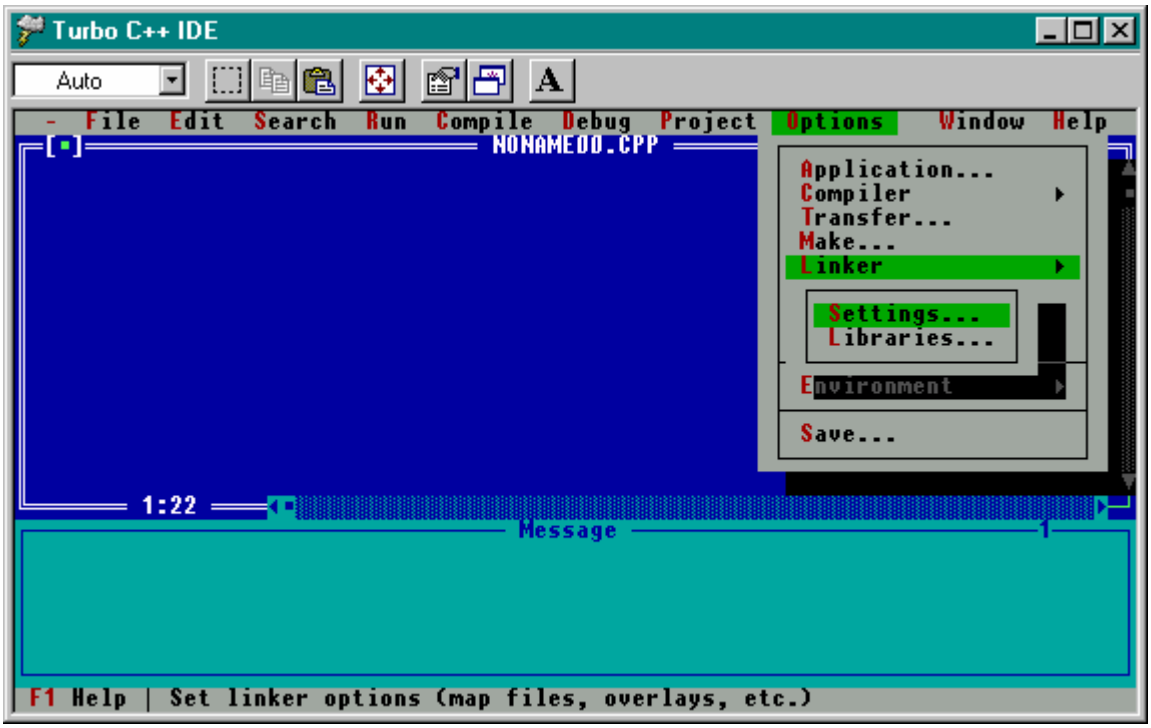


Figura 3.1 Menu Options

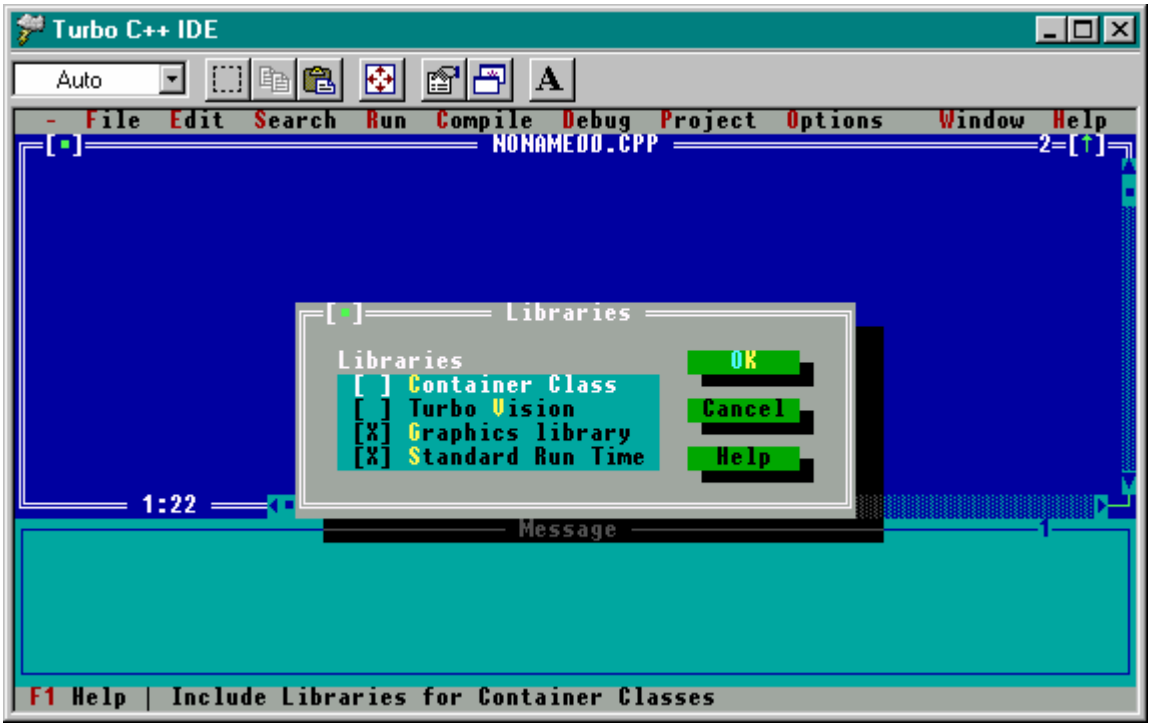


Figura 3.2 Librería gráfica habilitada

Ahora podemos compilar sin problemas algún programa de ejemplo, sin embargo todavía falta lo siguiente, los programas creados dependen de otros archivos para poderse ejecutar, estos son los archivos *.bgi, que son los controladores para el adaptador gráfico del cual se disponga, además si se incluyen fuentes se deben de contar con los archivos *.chr. La ruta por defecto donde se ubican estos archivos es la siguiente C:/TC/BGI.

Se puede disponer de estos archivos de dos formas, una es copiar los archivos a la carpeta donde tenemos el archivo ejecutable, y la segunda es cambiar nuestro directorio de trabajo a C:\TC\BGI en el menú FILE elegir CHANGE DIR

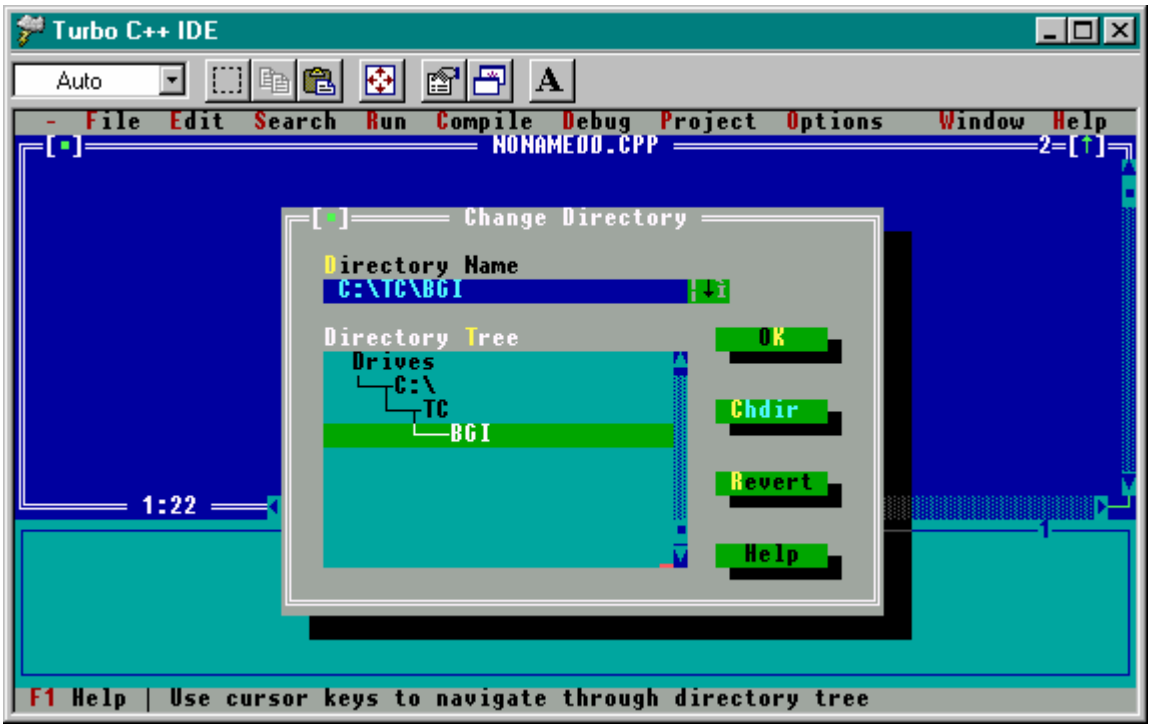


Figura 3.3 Configuración del directorio

De lo contrario nos aparecería un error como el siguiente

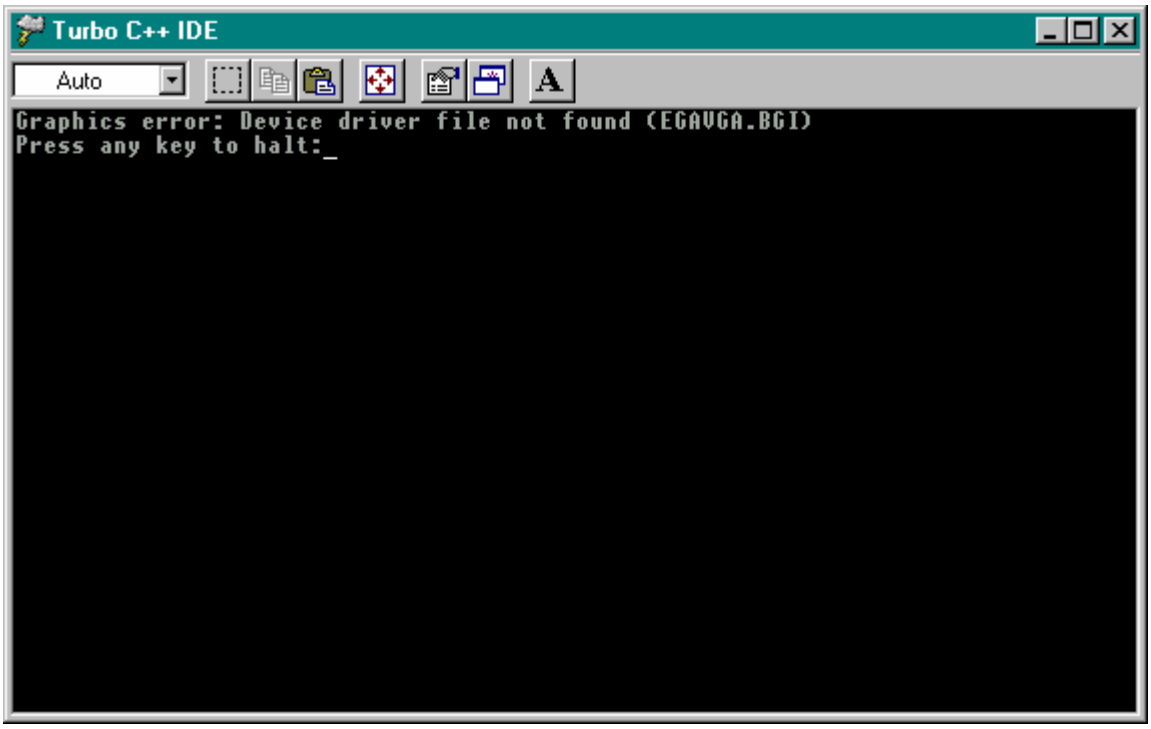


Figura 3.4 Error gráfico

Una vez sabiendo todo lo anterior podemos compilar y correr nuestros programas gráficos sin ningún problema, en nuestros programas debemos incluir la librería graphics.h, tal como el siguiente ejemplo:

```
                        Programa aplicgr.cpp
/*Programa aplicgr.cpp*/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   int radius = 100;
   int i, j;
   /* initialize graphics and local variables */
```

```
   initgraph(&gdriver, &gmode, “”);

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode ¡= grOk)  /* an error occurred */
   {
      printf(“Graphics error: %s\n”, grapherrormsg(errorcode));
      printf(“Press any key to halt:”);
      getch();
      exit(1); /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;
   setbkcolor(WHITE);

   setcolor(5);
   for(i=0; i<10;i++) line(10,(10*i), 100,100);

   settextstyle(4,0,4);
   setcolor(8);
   outtextxy(200,50,”APLICACIÓN GRAFICA”);

   j=0;
   for(i=0; i<15;i++){      setcolor(i);
                         j=j+10;
                         rectangle(20+j,20,50+j,30);
 }
   setcolor(BLUE);
   circle(midx, midy, radius);

   /* clean up */
   getch();
   closegraph();
   return 0;
}
```

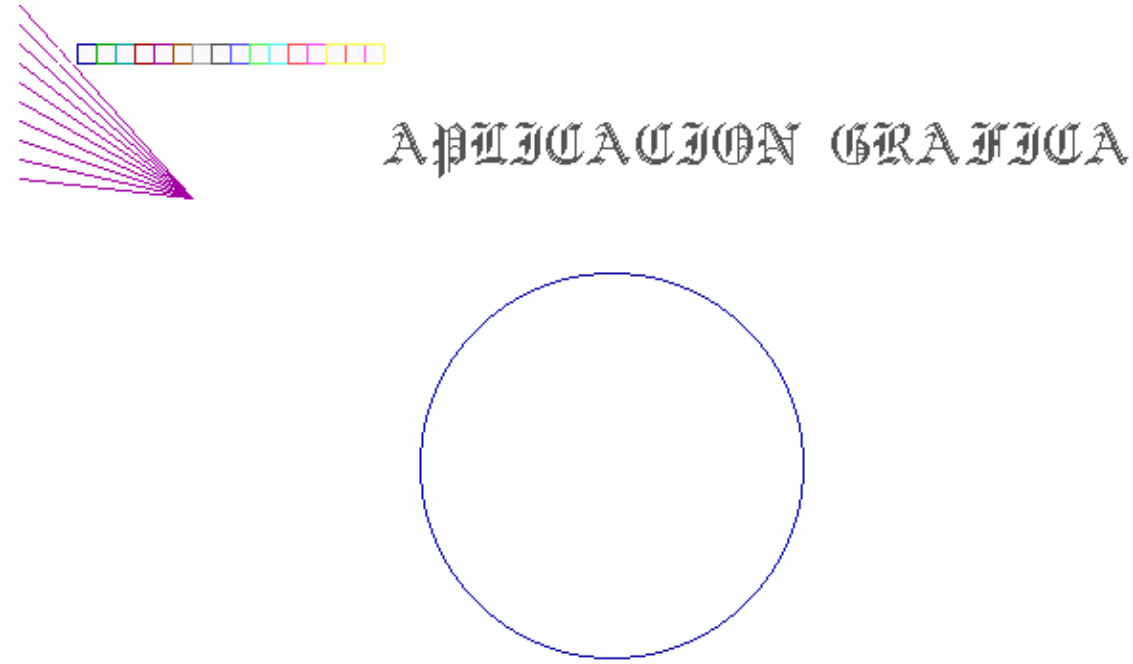


Figura 3.5 Salida del programa aplicgr.cpp

3.2.1 Modo texto y Modo gráfico

Tenemos que diferenciar los dos modos con los que contamos para desplegar nuestra información en la pantalla, el primero de ellos es el modo texto, el cual se compone de 25 renglones y 80 columnas, y el modo gráfico el cual puede variar dependiendo de la resolución que nos proporcione nuestro adaptador gráfico, junto con el tipo de monitor del cual dispongamos.

Para nuestro caso usaremos el modo de 640 x 480 pixels y 16 colores, mismo que es proporcionado por el controlador EGAVGA.bgi.

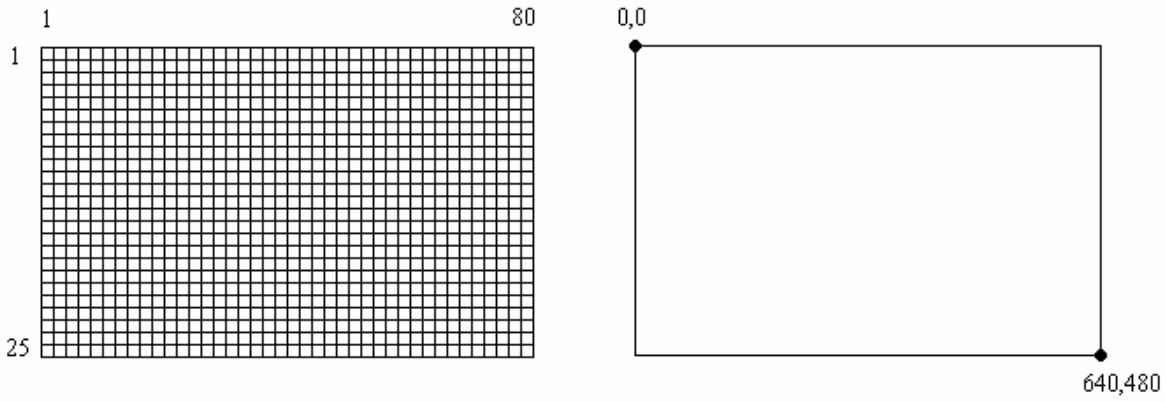


Figura 3.7 Modo texto y Modo Gráfico

En ambos modos el punto de origen es la esquina superior izquierda, hecho que deberá ser tomado muy en cuenta para desplegar correctamente nuestras gráficas.

3.2.2 Inicialización del modo gráfico

Tenemos dos opciones, la primera es dejar todo en manos de la autodetección (Automática) , misma que fue usada en el primer ejemplo, en ella se elige de modo automático la resolución y el número de colores, de esta forma siempre dispondremos del modo más alto posible que nuestro adaptador gráfico sea capaz de soportar.

La segunda es pasar los parámetros requeridos a las funciones de inicialización, para poder usar algún modo de vídeo en particular.

CONTROLADOR GRÁFICO	MODO GRÁFICO	VALOR	COLUMNA X RENGLÓN	PALETA	PÁGINAS
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 colores	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC1	2	320 x 200	C2	1
	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 colores	1
	MCGAHI	5	640 x 480	2 colores	1
EGA	EGALO	0	640 x 200	16 colores	4
	EGAHI	1	640 x 350	16 colores	2
EGA64	EGA64LO	0	640 x 200	16 colores	1
	EGA64HI	1	640 x 350	4 colores	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 colores	1 (64 K)
	EGAMONOHI	3	640 x 350	2 colores	2 (256 K)
HERC	HERCMONOHI	0	720 x 348	2 colores	2
ATT400	ATT400C0	0	320 x 200	C0	1
	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 colores	1
	ATT400HI	5	640 x 400	2 colores	1
VGA	VGALO	0	640 x 200	16 colores	2
	VGAMED	1	640 x 350	16 colores	2
	VGAHI	2	640 x 480	16 colores	1
PC3270	PC3270HI	0	720 x 350	2 colores	1
IBM8514	IBM8514HI	1	1024 x 760	256 colores	
	IBM8514LO	0	640 x 480	256 colores	

Tabla 3.1 Modos gráficos para cada controlador de vídeo.

Inicialización automática. Podemos valernos de cualquiera de los programas de ejemplo para poder inicializar nuestro modo gráfico.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
```

```
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }


/*NUESTRO CODIGO VA AQUI*/


/* clean up */
   getch();
   closegraph();
   return 0;
}
```

La línea siguiente contiene las variables requeridas como parámetro para la función initgraph() como podemos apreciar una de ellas se inicializa con el valor DETECT, lo cual hace que la autodeteccion se lleve a cabo, la ultima variable errorcode sirve para leer el estatus de inicialización dado por la función graphresult();

```
int gdriver = DETECT, gmode, errorcode;
```

La función para inicializar el modo gráfico es initgraph(); misma que requiere de tres parámetros

1.-Una variable para el controlador gráfico,

2.-Una variable para el modo gráfico del controlador

3.-Una cadena que indica la ruta del controlador gráfico

```
initgraph(&gdriver, &gmode, "");
```

Si el parámetro de la cadena esta vacío, o no se escribe nada, entonces initgraph(); busca en el directorio actual. La función graphresult(); nos indica el estado de la inicializacion, de ser correcto, el valor devuelto, debe ser igual a cero.

```
errorcode = graphresult();
```

A continuación si el código de error es diferente de cero (grok) entonces se despliega un mensaje de error correspondiente al valor de errorcode y se termina el programa.

```
if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
```

Por último tenemos que cerrar el modo gráfico y liberar la memoria ocupada.

```
closegraph();
```

Inicialización manual. Para inicializar de manera manual el modo gráfico debemos tomar en consideración a nuestro adaptador gráfico. Por ejemplo no podemos tener la resolución de 1024 x 760 que nos ofrece el modo IBM8514HI , si nuestro adaptador es del tipo VGA.

CONTROLADOR GRÁFICO (CONSTANTE)	VALOR
DETECT	0 (Valor para la autodetección)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Tabla 3.2 Controladores gráficos BGI

Para usar el controlador EGA, en el modo EGALO, se tiene lo siguiente:

```
int gdriver=EGA, gmode=EGALO;

initgraph(&gdriver, &gmode, "\\tc\\bgi");
```

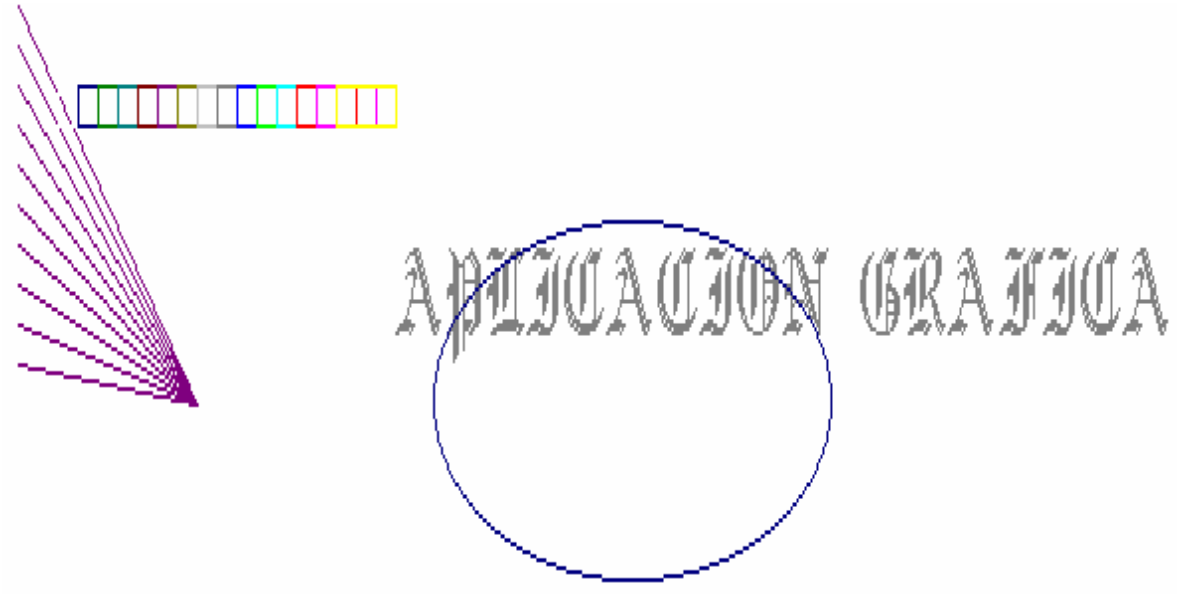


Figura 3.8 Salida del programa aplicgr.cpp en modo EGALO.

3.2.3 Despliegue de gráficos en pantalla.

El siguiente gráfico muestra las etapas para desplegar gráficos en la pantalla.

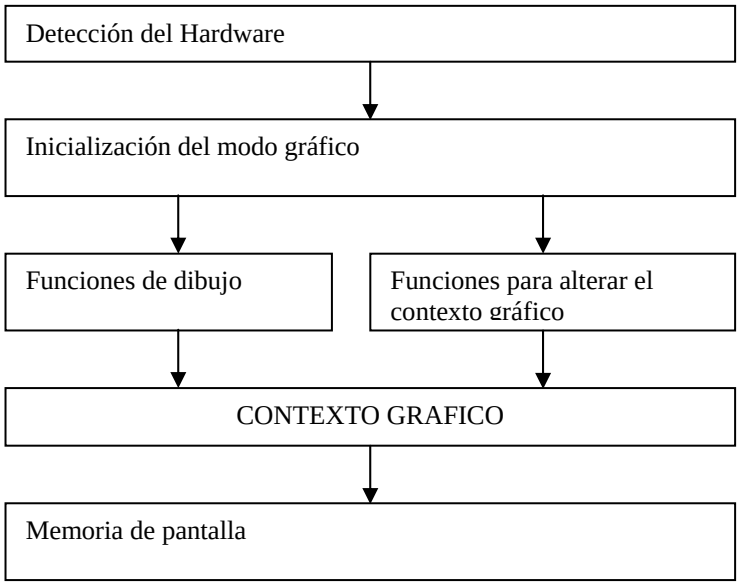


Figura 3.9 Despliegue de gráficos en la pantalla

Detección del hardware. Trata de averiguar que tipo de monitor y adaptador gráfico tenemos, en base a esto dispondremos de algún modo gráfico, o configuración.

Inicialización del modo gráfico. Una vez sabiendo las características de nuestro hardware, podremos trabaja con el, ya con alguna configuración especifica, dispondremos de una o mas resoluciones y de uno o más colores.

Funciones de dibujo. Son la funciones que nos permiten desplegar gráficos en a pantalla.

Funciones para alterar el contexto gráfico. El contexto gráfico es la serie de características propias de nuestros elementos básicos de dibujo, ya sea color, ancho, estilo, etc. No todas las características aplican a los mismos elementos gráficos.

Por ultimo, esta la memoria de la pantalla en la cual se despliegan los gráficos, dependiendo del adaptador dispondremos de una o más paginas para dibujar.

3.2.4 historia de los adaptadores gráficos

La primera tarjeta de vídeo, que se presentó junto al primer PC, fue desarrollada por IBM en 1980, recibió el nombre de **MDA** (*Monochrome Display Adapter*), sólo era capaz de trabajar en modo texto, representando 25x80 líneas en pantalla. Apenas disponía de RAM de vídeo (4 Kbytes) lo que hacía que sólo pudiera trabajar con una página en memoria. Para este tipo de tarjetas se usaban monitores monocromo (normalmente de tonalidad verde) de ahí el nombre que recibe esta tarjeta. Durante muchos años esta tarjeta, fue tomada como el estándar en tarjetas de vídeo monocromo.

Los primeros gráficos y colores llegaron a los ordenadores en 1981 con la **CGA** (*Color Graphics Adapter*), ésta era capaz de trabajar tanto en modo texto como en modo gráfico.

En modo texto al igual que la MDA, representa 25 líneas y 80 columnas en pantalla, pero el texto era menos legible, debido a que los diferentes caracteres se basaban en una

matriz de puntos más pequeña que en el caso de las tarjetas MDA. En modo gráfico, la CGA podía representar 4 colores con una resolución de 320 x 200 puntos.

La **CGA** estaba equipada con cuatro veces más memoria que su antecesora (16Kbytes) y podía conectarse a monitores RGB que eran capaces de emitir color (una mejora considerable si pensamos que la **MDA** se conectaba a monitores monocromo).

Dejando de lado por un momento a IBM, hablaremos de la **HGC** (*Hércules Graphics Card*) que salió al mercado un año después de la aparición del primer PC. Sus posibilidades con respecto de las anteriores eran abrumadoras puesto que además del modo texto, la tarjeta HGC puede gestionar dos páginas gráficas con una resolución de 720x348 puntos en pantalla. Con ello combina la estupenda legibilidad en modo texto de la MDA con las capacidades gráficas de la CGA, ampliando incluso la resolución, sin embargo la tarjeta HGC no era capaz de mostrar color por pantalla por lo que no llegó a estandarizarse como la CGA. La tarjeta Hércules tenía una memoria de 643 Kbytes y no era totalmente compatible con las tarjetas de IBM.

IBM en 1985 presentó la **EGA** (*Enhaced Graphics Adapter*), esta tarjeta era totalmente compatible con la **MDA** y la **CGA**. Con una resolución en el modo gráfico de 640x350 puntos, se podían representar 16 colores diferentes de una paleta de 64 colores, trabajando en modo texto con 25 filas y 80 columnas, donde cada carácter es representado con una matriz de 14x 8 puntos. También se aumentó la RAM de vídeo

hasta 256KB, para tener espacio para representar varias páginas gráficas (como hacía la HGC).

Estas cifras hacían ya posible que los entornos gráficos se extendieran al mundo PC (los Apple llevaban años con ello), aparecieron entonces *GEM* (Graphics Environment Manager - Gestor en Entorno Gráfico), que fue creado por Digital Research a comienzos de 1985

y *Windows* entre muchos.

Acercándonos al año 1990, IBM desarrollo dos tarjetas de vídeo, la **VGA** y la **MCGA,** esta última estaría destinada a los ordenadores PS/2 de gama baja que montaba la compañía.

La **MCGA** (*Memory Controller Gate Array*). En lo que se refiere a modo texto, estas tarjetas se comportan igual que una **CGA** con sus 25 x 80 caracteres en modo texto, donde se puede elegir el color de texto y fondo de una paleta de 16 colores. Al contrario que en la tarjeta CGA su resolución horizontal no es de 200 líneas, sino de 400 líneas, por lo que la definición de los caracteres es mucho mejor.

Las tarjetas **VGA** (*Video Graphics Array*) supusieron un nuevo paso en la consecución de gráficos de alta calidad, al representar *256* colores a escoger de una paleta de 262.144 tonalidades, con una resolución de 640 x 480 puntos en modo gráfico y 720x400 puntos en modo texto.

VGA es compatible con las tarjetas gráficas anteriores, **MDA**, **CGA** y **EGA**, de forma que el software desarrollado para las tarjetas gráficas anteriores puede ser utilizado sin problemas por un ordenador con tarjeta VGA. Las tarjetas VGA transmitían la señal al monitor en forma analógica. Esto hace que sólo funcionen con monitores analógicos VGA o con monitores multiscan, los cuales admiten señales tanto analógicas como digitales.

Las primeras tarjetas VGA tenían una memoria de pantalla de 256 Kbytes, pudiendo representar 16 colores en resolución 640 x 480 o 256 colores en resolución 350x200. Sin embargo, posteriormente aparecieron tarjetas VGA con mayor cantidad de memoria, pudiéndose representar un mayor número de colores en su más alta resolución. Otra particularidad es que cuando se enciende el ordenador, la tarjeta VGA detecta si el monitor que tiene conectado es color o monocromo, y en caso de que sea monocromo, convierte la información de color en tonos de grises, enviando estos tonos al monitor.

3.3 LA BIBLIOTECA GRAPHICS.H

Es una biblioteca de funciones que se incluye en la distribución del TC++ 3.0 de Borland y que nos permite hacer usos de gráficos en nuestros programas. La forma de usar estas funciones es bastante intuitiva, además se cuenta con una serie de ejemplos para cada una de las funciones gráficas misma que esta dentro de la ayuda del TC++ 3.0. Esta biblioteca contiene funciones para inicializar y cerrar el modo gráfico; funciones para dibujar gráficos y funciones para alterar el contexto gráfico. Son las siguientes:

arc	bar	bar3d
circle	cleardevice	clearviewport
closegraph	detectgraph	drawpoly
ellipse	fillellipse	fillpoly
floodfill	getarccoords	getaspectratio
getbkcolor	getcolor	getdefaultpalette
getdrivername	getfillpattern	getfillsettings
getgraphmode	getimage	getlinesettings
getmaxcolor	getmaxmode	getmaxx
getmaxy	getmodename	getmoderange
getpalette	getpalettesize	getpixel
gettextsettings	getviewsettings	getx
gety	graphdefaults	grapherrormsg
_graphfreemem	_graphgetmem	graphresult
imagesize	initgraph	installuserdriver
installuserfont	line	linerel
lineto	moverel	moveto
outtext	outtextxy	pieslice
putimage	putpixel	rectangle
registerbgidriver	registerfarbgidriver	registerbgifont
registerfarbgifont	restorecrtmode	sector
setactivepage	setallpalette	setaspectratio
setbkcolor	setcolor	setfillpattern
setfillstyle	setgraphbufsize	setgraphmode
setlinestyle	setpalette	setrgbpalette
settextjustify	settextstyle	setusercharsize
setviewport	setvisualpage	setwritemode
textheight	textwidth	

El archivo de cabecera graphics.h debe ser incluido en todos nuestros programas gráficos, ya que contiene las definiciones para las funciones gráficas a usar. Esto lo hacemos mediante la instrucción:

```
#include < graphics.h>
```

3.3.1 Funciones para dibujar

Nos permiten desplegar elementos gráficos básicos:

putpixel. Es la más elemental de todas. Nos permite colocar un pixel en la pantalla, tiene como parámetros las coordenadas (x, y)en donde colocaremos el pixel y el color con el que se dibujara. Su sintaxis es la siguiente:

putpixel(int x, inty, int color);

line. Dibuja una línea entre dos puntos, usando el color, estilo en curso y espesor , (x1, y1) es el punto de origen, y (x2, y2) es el punto final, su sintaxis es la siguiente:

line(int x1, int y1, int x2, int y2);

rectangle. Dibuja un rectángulo, usando el color, estilo en curso y espesor. Sus parámetros son el punto superior izquierdo (x1, y1) y el punto inferior derecho(x2, y2). Su sintaxis es la siguiente.

rectangle(int x1, int y1, int x2, int y2);

bar. Dibuja una barra rectangular de dos dimensiones, la barra el rellenada usando el patrón de relleno y color de relleno en curso. Sus parámetros son el punto superior izquierdo (x1, y1) y el punto inferior derecho(x2, y2). Su sintaxis es la siguiente.

bar(int x1, int y1, int x2, int y2);

circle. Dibuja un circulo, usando el color en curso. Tiene por parámetros el punto central (x, y) y el radio del circulo. Su sintaxis es la siguiente:

circle(int x, int y, int radio);

arc. Nos permite dibujar un arco circular los parámetros requeridos son el punto central (x, y), el ángulo de inicio, el ángulo final y el radio. Su sintaxis es la siguiente:

arc(int x , int y, int ang_ini, int ang_fin, int radio);

ellipse. Dibuja un arco elíptico, usando el color en curso. Sus parámetros son el centro de la elipse (x, y), el ángulo inicial(ang_ini) , el ángulo final (ang_fin), el radio en el eje *x*, y el radio en el eje *y*. Su sintaxis es la siguiente:

ellipse(int x, in y, int ang_ini, int ang_fin, int xradio, int yradio);

fillellipse. Dibuja una elipse rellena usando el color y patrón de relleno en curso. Sus parámetros son el centro de la elipse (x, y), el ángulo inicial(ang_ini) , el ángulo final (ang_fin), el radio en el eje *x*, y el radio en el eje *y*. Su sintaxis es la siguiente:

fillellipse(int x, in y, int ang_ini, int ang_fin, int xradio, int yradio);

3.3.2 Funciones para el contexto gráfico.

Nos permiten manipular nuestro contexto gráfico.

setcolor. Establece el color en para dibujar, el color esta en un rango de 0 a getmaxcolor(), para el caso del controlador EGAVGA.bgi, el número de colores es 16. Se emplea como argumento el número ó el nombre del color en la siguiente lista:

Constante	Valor
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

Tabla 3.3 Colores soportados por el controlador EGAVGA

Su sintaxis es la siguiente:

setcolor(int color);

setlinestyle. Esta función se usa para establecer el estilo de línea que se desea puede ser línea continua, línea punteada, línea interrumpida, o un patrón de bits definidos por el usuario. Esta función requiere tres parámetros, que son:

1) Tipo de línea: Puede ser SOLID_LINE, DOTTED_LINE, CENTER_LINE, DASHED_LINE o USERBIT_LINE .
2) Patrón: Este argumento es ignorado a menos que se utilice un patrón de bits definido por el usuario.
3) Ancho de línea: Define la amplitud en pixels, solo esta definido para los valores 1 y 3 (NORM_WIDTH y THICK_WIDTH).

Su sintaxis es la siguiente:

setlinestyle(int tipo, int patron, int ancho);

setfillstyle. Establece el patrón de relleno y el color de relleno en curso. Su sintaxis es la siguiente

setfillestyle(int patrón, int color);

El patrón de relleno puede ser uno los siguientes en la lista:

Nombre	Valor	Relleno con
EMPTY_FILL	0	Color de fondo
SOLID_FILL	1	Relleno sólido
LINE_FILL	2	Relleno con líneas
LTSLASH_FILL	3	///////
SLASH_FILL	4	/////// grueso
BKSLASH_FILL	5	\\\\\\\ grueso
LT BKSLASH_FILL	6	\\\\\\\
HATCH_FILL	7	Tramado ligero
XHATCH_FILL	8	Tramado cruzado
INTERLEAVE_FILL	9	Líneas entrelazadas
WIDE_DOT_FILL	10	Puntos espaciados
CLOSE_DOT_FILL	11	Puntos cerrados
USER_FILL	12	Definido por el usuario

Tabla 3.4 Patrones de relleno

Como ya se menciono se cuenta con ayuda dentro de turbo C++ 3.0. Usando la combinación de teclas CTRL+F1 aparece un listado, de donde podemos elegir un tema mediante el teclado, solo basta escribir por ejemplo el nombre de laguna función existente. Si estamos editando un programa, podemos colocarnos sobre alguna función o palabra clave y usar la misma combinación de teclas, lo cual dará por resultado que se despliegue la ayuda en una ventana para esa función o palabra clave en especifico.

3.4 PROGRAMACIÓN DEL RATÓN

Al igual que con otros periféricos, hay en la BIOS una interrupción que se encarga de manejar este dispositivo. Son una serie de funciones que normalmente acompaña en un disco al ratón, y que lo cargamos mediante un archivo que queda

residente en memoria (mouse.com, mouse.sys...). Esta interrupción es la 33h, la cual nos ofrece un surtido de funciones para que podamos manejar el ratón, desde comprobar tipo de ratón, botones, hasta poder cambiar la velocidad a la cual queremos que vaya el ratón (sensibilidad) y limitar las coordenadas por donde queremos que se mueva en pantalla. Para manejar el ratón tenemos dos posibilidades, y como casi siempre, la más eficaz es la más complicada.

Estos son el control del ratón mediante el método polling y el control mediante interrupción. Mediante polling significa que cada vez que queramos saber las coordenadas del ratón o que botones hay pulsados tendremos que llamar a las funciones de la interrupción, pero es muy lento, pues tenemos que estar "mirando" todo el tiempo al ratón. Mediante el segundo método esto cambia. Un controlador que instalaremos se encargará de ir actualizando unas variables que indicarán en cada momento la posición del ratón y si existe algún botón pulsado. Solamente las actualizará cuando detecte algún movimiento del ratón, de manera que el ahorro de recursos del sistema con respecto al método polling es considerable.

3.4.1 Las interrupciones

Son señales enviadas a la CPU para que termine la ejecución de la instrucción en curso y atienda una petición determinada, continuando más tarde con lo que estaba haciendo. Una interrupción es una pequeña subrutina que instala un programa o un

sistema. Establecer una subrutina como interrupción permite que otros programas puedan utilizarla simplemente llamándola.

En una PC existen 256 interrupciones, enumeradas desde 00 a FF (255 en hexadecimal). Muchas de estas interrupciones son utilizadas por el BIOS y por DOS para brindar a las aplicaciones diversas funciones (Por ejemplo, el BIOS llama a INT 09 cada vez que se pulsa una tecla, INT 10 es utilizado por BIOS para dar a las aplicaciones que la llamen servicios de vídeo, a INT 1C 18.2 veces por segundo e INT 21 es utilizado por DOS para brindar sus servicios como abrir archivos, utilizar carpetas, cambiar la hora, etc.; INT 33 brinda todos los servicios para utilizar el ratón, entre otros).

Muchos de los servicios de interrupciones están hechos para que cualquier aplicación pueda utilizarla. Para utilizarla debemos recurrir a los registros básicos del CPU (AX, BX, CX, DX, etc). Como manera de convención se establece que con AX se especifica el valor de la función que se desea utilizar, los demás registros se usarán de acuerdo a los requerimientos de la función a llamar. Para poder utilizar una función debemos saber en qué interrupción está alojada, qué valor de AX la identifica dentro de esa interrupción y qué parámetros son utilizados (Si utiliza BX, CX, DX, etc.).

Tanto el valor de la interrupción como el valor utilizado en AX se brindan en formato hexadecimal como convención, así que cuando leas información de una interrupción en particular y ves valores en AX se debe comprender que están en hexadecimal a menos que se indique otro formato numérico.

Como forma de ejemplo utilizaremos funciones de la INT 33 (Interrupción del ratón o "Mouse") y nombraremos algunos valores de AX que corresponderán a las funciones disponibles.

INT 33 AX = 1 Hace visible el puntero del ratón

INT 33 AX = 2 Oculta el puntero del ratón

3.4.2 El Ratón.

El ratón se controla normalmente a través de llamadas a la INT 33h. Existen toda suerte de funciones para controlar su posición, el estado de los botones, el puntero que se visualiza... todas ellas son bastante intuitivas y aptas para un programador en lenguajes de alto nivel.

El funcionamiento se establece a partir de interrupciones de puerto serie. Se trasmiten 3 bytes cada vez que hay un envío: en ellos se indica cuánto se ha movido el ratón en los ejes X e Y desde la última vez, así como el estado de los botones. La unidad de medida, son los *Mickeys*, que según la resolución del aparato serán 1/200 ó 1/400 pulgadas.

Los desplazamientos se toman en complemento a dos; como hay 8 bits por cada eje, el movimiento puede oscilar en el rango +128 a -127. Hay además un bit por cada

botón. De los 7 bits recibidos en cada interrupción, el más significativo (bit 6) está a 1 en el primer envío y a 0 en los restantes, con objeto de evitar malas interpretaciones de la secuencia si se pierde alguna interrupción por cualquier motivo.

A continuación se muestran todas las funciones que sirven para programar el ratón. Todas estas funciones para la programación del ratón son las mismas en todos los lenguajes.

Función 0: Inicialización y estado del ratón.
Función 1: Visualiza el puntero.
Función 2: Oculta el puntero.
Función 3: Obtiene la posición del ratón y el estado de los botones.
Función 4: Define la nueva posición del puntero del ratón.
Función 5: Obtiene la información sobre los botones pulsados.
Función 6: Obtiene la información sobre los botones libres.
Función 7: Define las posiciones horizontales máxima y mínima del puntero.
Función 8: Define las posiciones verticales máxima y mínima del puntero.
Función 9: Define la forma del puntero en modo gráfico.
Función 10: Define los atributos y el color del puntero en modo texto.
Función 11: Lee los contadores del movimiento del ratón.
Función 12: Define la dirección y la máscara de la llamada a la subrutina de interrupciones.
Función 13: Activa la emulación del lápiz óptico.
Función 14: Desactiva la emulación del lápiz óptico.
Función 15: Define la relación entre mickeys y pixels.
Función 16: Desactivación condicional.
Función 19: Define el umbral de doble velocidad.
Función 20: Intercambia las rutinas de interrupciones.
Función 21: Obtiene las necesidades de almacenamiento para el estado de las rutinas de control del ratón.
Función 22: Guarda el estado de las rutinas de control del ratón.
Función 23: Recupera el estado de las rutinas de control del ratón.
Función 24: Define la dirección y máscara de llamada de la subrutina alternativa.
Función 25: Obtiene la dirección de la interrupción de usuario alternativa.
Función 26: Define la sensibilidad del ratón.
Función 27: Obtiene la sensibilidad del ratón.
Función 28: Define la frecuencia de interrupciones del ratón.
Función 29: Define el número de página del controlador gráfico.

Función 30: Obtiene el número de página del controlador gráfico.
Función 31: Desactiva las rutinas de control del ratón.
Función 32: Activa las rutinas de control del ratón.
Función 33: Inicialización por software.
Función 34: Define el lenguaje de los mensajes.
Función 35: Obtiene el número del lenguaje.
Función 36: Obtiene el número de versión de las rutinas de control, el tipo de ratón y el número de IRQ.
Función 37: Obtiene información general sobre las rutinas de control.
Función 38: Obtiene las coordenadas virtuales máximas.
Función 39: Obtiene las máscaras de puntero/pantalla y el contador de mickeys.
Función 40: Define la modalidad de vídeo.
Función 41: Enumera las modalidades de vídeo
Función 42: Obtiene el punto caliente del puntero.
Función 43: Carga las curvas de aceleración.
Función 44: Lee las curvas de aceleración.
Función 45: Obtiene/define la curva de aceleración activa.
Función 47: Inicialización del hardware del ratón.
Función 48: Define/obtiene información sobre el BallPoint.
Función 49: Obtiene las coordenadas virtuales máxima y mínima.
Función 50: Obtiene las funciones avanzadas activas.
Función 51: Obtiene la configuración de interruptores.
Función 52: Obtiene MOUSE.INI

3.4.2 Funciones básicas del ratón.

Estas funciones fueron encontradas en el archivo de ayuda creado por Cristóbal Tello, mismo que se puede encontrar en **www.lawebdelprogramador.com** y se muestra a continuación:

Función 00h (Inicializar el ratón. Saber si existe o no ratón)

Está función retorna 0 si no existe ningún ratón instalado. Si existe un ratón retorna el número de los botones de nuestro ratón.

Registro de entrada: AX=0000h Inicializa el ratón. (Reset)

Registros de salida: AX=FFFFh. Existe un ratón instalado.

BX=Nºde botones.

```
int MirarRaton()
{
  asm xor ax, ax
  asm int 33h
  asm cmp ax, -1
  asm je  ExisteRaton
  return 0;

ExisteRaton:
return _BX;
}
```

Función 01h (Mostrar ratón)

Esta función es muy sencilla, únicamente le pasamos el valor 01h al registro AX y el puntero del ratón se nos mostrará en el monitor. Si estamos en modo texto, el puntero del ratón tendrá forma cuadrada (como si de un cursor se tratase), en cambio, si nos encontramos en modo gráfico, el puntero será la típica flecha.

Registro de entrada: AX=0001h Mostrar el puntero del ratón.

```
int MostrarPuntero()
{
  asm mov ax, 01h
  asm int 33h
}
```

Función 02h (Ocultar el puntero del ratón)

Exactamente igual que en la anterior con la diferencia en que el puntero no se verá.

Registro de entrada: AX=0002h Ocultar el puntero del ratón.

```
int OcultarPuntero()
{
   asm mov ax, 02h
   asm int 33h
}
```

La función 03H (Posición el cursor y estado de los botones

Función 03h (Posición del puntero del ratón, estado de los botones)

Registro de entrada: AX=0003h

Registros de salida: CX=Posición horizontal

DX=Posición vertical.

BX=Nº de botón pulsado.

Los registro CX y DX nos mostrarán las coordenadas horizontales y verticales respectivamente. Hay que tener en cuenta en que modo gráfico estamos porque a veces no corresponderán realmente. Si estamos en el modo gráfico 13h (300x200), bastante popular en el mundillo de las Demos, tendríamos que dividir el registro CX, (posición horizontal) entre 2. En cambio, si estamos en el modo texto las coordenadas (CX y DX) las tendremos que dividir por 8. Al hacer esto los valores que retornan estas variables siempre 0 para la columna y fila 1, 1 para la fila y columna 2 y así sucesivamente. Podemos "arreglar" esto haciendo que al resultado de la función se le sume 1.

Para saber qué botón está pulsado miraremos el registro BX. Si vale 0 no habrá ningún botón apretado. Si vale 1 el botón izquierdo estará apretado y si vale 2 estará el botón derecho apretado. Si están los botones izquierdo y derecho apretados el registro BX devolverá el valor 3.

Modo Texto

```
int PosVertTexto()
{
 asm mov ax, 03h
 asm int 33h
 return _CX/8
}

int PosHoriTexto()
{
 asm mov ax, 03h
 asm int 33h
 return _DX/8
}
```

Modo Gráfico

```
int PosVertGraf()
{
  asm mov ax, 03h
  asm int 33h
  return _CX / 2
}

int PosHoriGraf()
{
  asm mov ax,03h
  asm int 33h
  return _DS;
}
```

Y para saber el/los botón/es pulsados.

```
int BotonPulsado()
{
 asm mov ax, 03h
 asm int 33h
 return _BX;
}
```

La función 04H (Posicionar el ratón)

Función 04h (Posicionar el puntero del ratón)

Registro de entrada: AX=0004h

CX=Posición horizontal

DX=Posición vertical.

Colocará el cursor en las coordenadas que le digamos. Según en que modo estemos trabajando debemos transformarlas.

La función 07H y 08H

Funciones 07h y 08h (Limitar los movimientos de puntero del ratón)

Registro de entrada: AX=0007h

CX=Mínimo valor para el eje horizontal

DX=Máximo valor para el eje horizontal.

Registro de entrada: AX=0008h

CX=Mínimo valor para el eje vertical

DX=Máximo valor para el eje vertical

Estas funciones hacen que se limite el ratón. Nos puede servir para cuando pongamos alguna ventana y queremos que el puntero del ratón no salga de ella. Debemos convertir las coordenadas al modo que estamos utilizando.

La función 09H (Cambiar el puntero)

Función 09h (Cambiar el cursor)

Registro de entrada: AX=0009h

DX=Dirección OFFSET donde esta el cursor

ES = Segmento bitmap donde esta el cursor del ratón.

He aquí un ejemplo que genera un cursor en forma de T.

```
void mouse_curs()
{
  union REGS regs;

  struct SREGS sregs;
  int mask[36] = {0xffff, 0x8001, 0x8001, 0x8001,0xf81f, 0xf81f, 0xf81f, 0xf81f,0xf81f,
0xf81f, 0xf81f,     0xf81f,0xf81f, 0xf81f, 0xf81f, 0xffff,0x0, 0x7ffe, 0x7ffe, 0x7ffe,
0x7e0, 0x7e0, 0x7e0, 0x7e0, 0x7e0,   0x7e0, 0x7e0, 0x7e0, 0x7e0, 0x7e0, 0x7e0,
0x0,0x7e0, 0x7e0, 0x7e0, 0x0 };
  regs.x.ax = 0x09;
  regs.x.bx = 0x4;
  regs.x.cx = 0xb;
  regs.x.dx = FP_OFF((void far *)&mask[0]);
  sregs.es  = FP_SEG((void far *)&mask[0]);
  int86x(0x33, &regs, &regs, &sregs);
}
```

CAPITULO 4

DESARROLLO DE LA INTERFAZ GRÁFICA

4.1 INTRODUCCIÓN

Si bien no se sigue un método particular para el diseño de esta interfaz gráfica, se propone una serie de pasos, que nos ayuda a identificar los problemas o necesidades, para después trabajar sobre ellas, pese a cualquier metodología que se emplee, el programador es el que decide como resolverá el problema. El buscar otras alternativas nos trae ventajas así como desventajas, por una parte no estamos atados a pasos predeterminados y nuestra creatividad florecerá enormemente, pero podemos perder de vista rápidamente de nuestro objetivo y los módulos pueden no ser compatibles. En este capítulo se muestra como evolucionó la interfaz, los programas de prueba, y su aplicación final.

4.2 METODOLOGÍA.

Se dice que la programación consiste en un 1% de inspiración y 99% de depuración, la frase anterior bien puede definir el desarrollo de este programa. Los pasos a seguir en el desarrollo de esta interfaz son los siguientes:

1. Preguntar que recursos necesitamos para desarrollar nuestro programa, para después reunirlos, analizarlos, y emplearlos.
2. Definir las partes componentes del programa, es decir hacer una lista de las funciones que deberá llevar a cabo

3. Desarrollar cada una de las partes componentes del programa. Se trabaja cada una de las partes por separado, cuidando de que exista una forma de acoplar las partes. Se aconseja plasmar las ideas sobre papel, para después trabajas sobre ellas.
4. Ensamblar las partes componentes del programa. Es mejor si se ensamblan las partes componentes del programa a medida de que se terminen.

Esta serie de pasos se debe aplicar cada vez que origine algún nuevo problema por resolver.

4.2.1 Recursos necesarios.

¿Qué se necesita para desarrollar un simulador de compuertas? Es la pregunta obligada, en primer lugar tenemos que contar con los conocimientos básicos en algún lenguaje de programación que nos permita desplegar gráficos en la pantalla. Los capítulos 2 y 3 nos dan estos recursos, que básicamente son dos:

a) Conocimiento del lenguaje.
b) Despliegue de gráficos en la pantalla.

Estos dos recursos definen la parte gráfica del programa, pero existe una parte importante que es la parte funcional y lógica, es decir podemos por ejemplo dibujar un diagrama lógico con compuertas, pero la parte importante es hacer las asociaciones

lógicas y relacionar los elementos gráficos para que en realidad lleven a acabo una función, que es resolver un diagrama lógico. Esta parte es la más difícil de visualizar.

¿Cómo controlar los elementos gráficos?, la respuesta no es sencilla, y este problema es el que nos llevará más trabajo, sin embargo una pieza clave a la solución, es el tener conocimiento del elemento gráfico, es decir conocer sus atributos, posición, etc. , por ejemplo:

Elemento	**Atributos**
Pixel	Color, punto que lo definen en la pantalla P(x, y).
Línea	Color, punto inicial (x1,y1), punto final (x2, y2).

Tabla 4.1 Atributos de elementos gráficos

Los atributos que tiene algún elemento gráfico pueden ser almacenados, modificados y asociados por medio de variables, este principio puede ser aplicado a elementos más complejos, por ejemplo una compuerta, considerando únicamente el dibujo de la compuerta, tendrá:

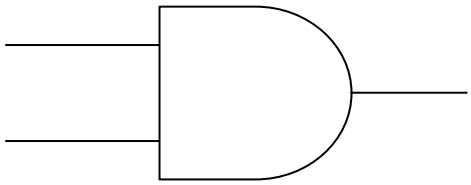


Figura 4.1 Compuerta AND

Elementos gráficos de la compuerta
Punto inicial desde donde se dibuja la compuerta
Tipo de compuerta (AND, OR, EXOR, etc.)
Color con el cual se dibuja

Figura 4.2 Atributos gráficos de una compuerta

Ahora mencionaremos los elementos lógicos de la compuerta, los cuales están formados por sus terminales y la función que realiza.

Elementos lógicos de la compuerta
Entrada 1
Entrada 2
Salida
Función lógica a realizar

Figura 4.3 Atributos lógicos de una compuerta

Se pueden proponer más atributos tanto gráficos como lógicos, entre mayor sea el numero de estos, mayor será el control que tendremos sobre nuestros elementos gráficos. Durante el desarrollo de las funciones componentes, veremos como trabajar los atributos de nuestros elementos, los cuales pasaran a ser variables que podemos asociar y manipular.

4.2.2 Partes componentes del programa

Para definir las funciones que componentes el programa, se tomó en consideración funciones vistas en simuladores comerciales, de las muchas posibles se eligieron algunas cuantas, de manera que se contara con las funciones que nos permitieran llegar a nuestro objetivo, “Resolver diagramas lógicos”.

En base a los recursos reunidos, se decidió crear una interfaz de comandos, es decir acceder a las funciones del programa, mediante instrucciones desde el teclado, ya

que es una interfaz sencilla de implementar. De forma casi paralela, se trabajo con una interfaz basada en el uso del ratón, la cual no se completo, debido a la falta de experiencia con este tipo de interfaz, ya que era necesario darle un nuevo enfoque al problema y replantear todo de nuevo.

El siguiente diagrama, nos muestra los bloques componentes del programa, que se pensaron en un principio.

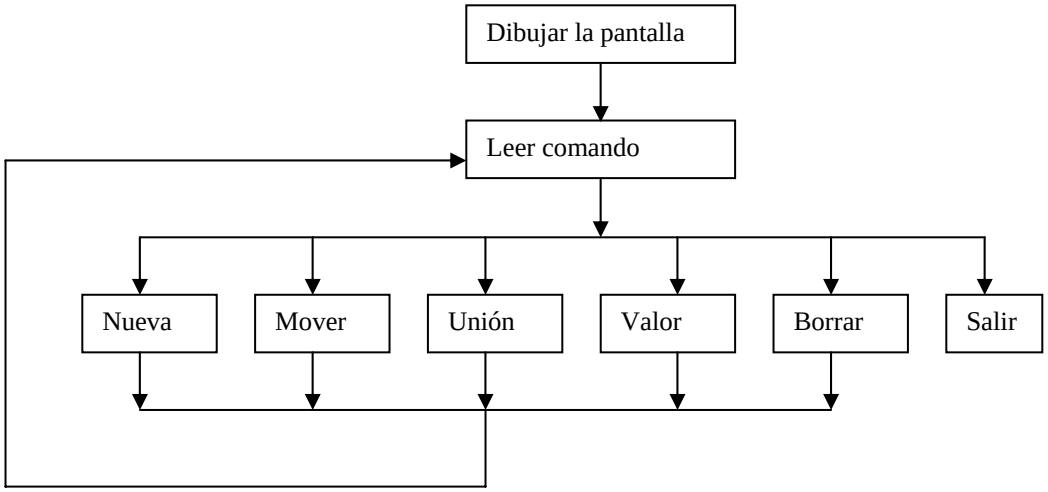


Figura 4.2 Diagrama de flujo inicial

De estos bloques se elimino ***Borrar*** para hacer la interfaz más sencilla, también debido a que se puede prescindir de esta función, los bloques ***Unión*** y ***Valor*** se juntaron en uno solo ***Valores***, y claro falta la función ***Resultado***, quedando las funciones como siguen:

- Dibuja pantalla. Dibuja un menú con los comandos disponibles.

- Leer comando. Espera por un comando y lo valida.

- Nuevo. Dibuja una compuerta y guarda sus atributos

- Mover. Permita seleccionar y mover una compuerta.

- Valores. Se asigna un valor a la terminal de una compuerta ó une una terminal con otra.

- Resultado. Resuelve el diagrama lógico.

- Salir. Sale del programa.

4.3. MOVIMIENTO

Una de las principales preocupaciones sobre del programa, fue como mover un objeto por la pantalla de manera interactiva, y este problema es el que tratamos primero, debido al impacto visual que provoca.

4.3.1 Movimiento de objetos.

En la siguiente dirección electrónica http://www.cworld.nu se localizaba el programa pelota.c, dicha dirección ya no existe, ahora se puede encontrar el programa en http://www.solocodigo.com , mismo que se muestra a continuación.

```
/*********************************************************************
*
 Nombre: PELOTA.C
 Utilidad: Muestra la utilizaci¢n de los cursores, la tecla INTRO y
ESC.
 Autor: Sergio Pacho    Fecha: 27 de septiembre de 1998

 Este programa es una sencilla demostración de como utilizar los
cursores
 para desplazar un objeto por la pantalla. Lo único que te pido es que
no
 modifiques los créditos, y que se lo pases a todos tus amigos,
colegas..

 También puedes decir que lo has conseguido en:
 http://www.solocodigo.com
*********************************************************************
*/

#include <dos.h>
#include <graphics.h>

char t;
void main(void)
{
      int tarj,modo,x=320,y=240,c=1;
      unsigned int segment, offst;
      detectgraph(&tarj,&modo);
      initgraph(&tarj,&modo,"c:\\tc\\bgi");
      setcolor(0);
      setfillstyle(1,c);
      do
      {
            t=inport(0x60); /* Leemos la entrada del teclado */
            switch (t)
            {
                  case 28: /* Si se ha pulsado INTRO */
                        if (c<15) c++;
                        else c=1;
                        setfillstyle(1,c); /* Cambiamos el color del
c¡rculo */
                        break;
                  case 72: /* Cursor arriba */
```

```
                        y--; break;
                  case 75:
                        x--; break; /* Cursor izquierda */
                  case 77:
                        x++; break; /* Cursor derecha */
                  case 80:
                        y++; break; /* Cursor abajo */
            }
            fillellipse(x,y,40,40); /* dibujamos el c¡rculo */

            /* Esta es una forma pr ctica de borrar la pantalla sin
excesivos
                  parpadeos */
            *(char far *) MK_FP(0x40,0x1a)=*(char far *)
MK_FP(0x40,0x1c);
      }while(t!=1); /* Mientras no se pulse ESC */
}
```

Lo que podemos resaltar de este programa es que nos muestra la manera de controlar las flechas del teclado y por ende saber el código de otras teclas, como se muestra en el siguiente programa:

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

/* Armando Nava Linares */
/* Teclado.cpp*/
/* Programa para identificar la lectura del teclado*/

void main(void)
{
 int i;
 char t;
        for(i=0;i<20; i++){
            printf("\n Teclee un caracter");
            getch();

            t=inport(0x60);

        printf("\n El valor de T es: %d",t);

        }
}
```

Este programa nos solicita la entrada de un carácter, para después imprimir el valor de la tecla presionada, hay que notar que la variable *t* es declarada como tipo char, sin embargo desplegaremos esta variable como entera, para conocer el valor numérico leído desde el puerto asignado al teclado.

4.3.2 Creación de objetos.

Para dibujar objetos podemos agrupar sus partes componentes líneas, pixels, arcos, etc., en una sola función, por ejemplo, si queremos dibujar el numero 1 con líneas se tendrá algo como lo siguiente:

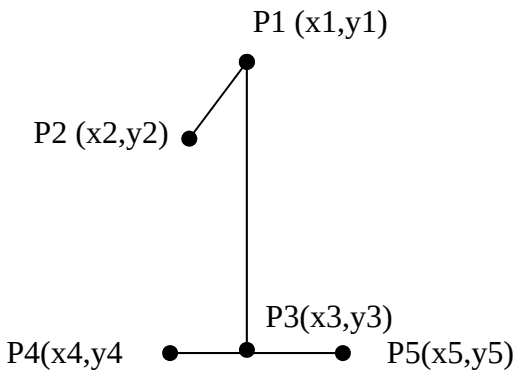


Figura 4.2 Representación del una figura con líneas

Este método requiere tres líneas, delimitadas por los puntos P1, P2, P3, P4 y P5, una función para dibujar este objeto, podría ser la siguiente:

```
int numero_uno(){
      line(x1,y1,x2,y2);
      line(x1,y1,x3,y3);
      line(x4,y4,x5,y5);
return(0);
}
```

Otra opción es usar un array, donde cada elemento representa a un pixel, utilizaremos esta segunda opción para dibujar nuestras compuertas. El array, almacena los datos de las compuertas en forma de unos y ceros, después se la función putpixel() para dibujarla.

```
void nand(int x,int y)  //Esta función requiere los parámetros x,y, que
es el punto inicial u origen
  {
   int i,j;
            //Array que contiene la información a graficar

int c_nand[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                    1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,
                    0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                    1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

};


  for(j=0;j<25;j++)
       {

      for(i=0;i<11;i++)
          {

           if(c_nand[i][j]==0)

putpixel(x+j,y+i,0); //coloca un pixel negro cuando encuentra un cero

           else

putpixel(x+j,y+i,15);  //de lo contrario coloca un pixel blanco
          }

      }

}
```

El siguiente programa nos permite seleccionar entre dos objetos creados por nosotros y mover a uno de ellos por medio del teclado (datos.cpp).

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
//datos.cpp
int cuad_2(int x,int y);
int cuad_1(int x,int y);

struct square{  int x,y;      //COORDENADAS
            int tipo;     //TIPO
            int numero;   //NUMERO

                    }cuadro[2];

int main(void)
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   int x,y;
   char kyb,t;
   int tipo;
   int aux;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

            cuadro[0].tipo=cuad_1(100,100);
            cuadro[0].x=100;
            cuadro[0].y=100;
            cuadro[0].numero=1;

            cuadro[1].tipo=cuad_2(200,200);
            cuadro[1].x=200;
            cuadro[1].y=200;
```

```
            cuadro[1].numero=2;

   outtextxy(20,400,"Que cuadro desea mover");
     getch();

    kyb=inport(0x60);


       switch(kyb)  {


case 2:        x=cuadro[0].x;
           y=cuadro[0].y;

              do
              {
              t=inport(0x60);  /*entrada desde el teclado */
              getch();
           switch(t)
           {

                  case 72: /*Cursor arriba */
                        y--;
                        break;
                case 75:
                        x--;
                        break; /*Cursor izquierda*/
                  case 77:
                        x++;
                        break; /*Cursor derecha */
                  case 80:
                        y++;
                        break; /*Cursor abajo */
     }
                   cleardevice();
                   cuadro[0].x=x;
                   cuadro[0].y=y;
                   aux=cuad_1(x,y);

                   aux=cuad_2(cuadro[1].x,cuadro[1].y);
     //     *(char    far    *)    MK_FP(0x40,0x1a)=*(char    far    *)
MK_FP(0x40,0x1c);

     }while(t!=1); /*Mientras no se pulse esc */

case 3:        x=cuadro[1].x;
             y=cuadro[1].y;

              do
              {
              t=inport(0x60);  /*entrada desde el teclado */
              getch();
           switch(t)
           {
```

```
                case 72: /*Cursor arriba */
                      y--;

                      break;
                case 75:
                      x--;

                      break; /*Cursor izquierda*/
                case 77:
                      x++;

                      break; /*Cursor derecha */
                case 80:
                      y++;
                      cleardevice();
                      break; /*Cursor abajo */
    }

                 cleardevice();
                 cuadro[1].x=x;
                 cuadro[1].y=y;
                 aux=cuad_2(x,y);

                 aux=cuad_1(cuadro[0].x,cuadro[0].y);
       //   *(char    far    *)    MK_FP(0x40,0x1a)=*(char    far    *)
MK_FP(0x40,0x1c);

     }while(t!=1); /*Mientras no se pulse esc */



 }


   /* clean up */
   getch();
   closegraph();
   return 0;
}


int cuad_1(int x, int y)  { int tipo_1=1;
                     int i,j;
                     int C_1[10][10]={1,1,1,1,1,1,1,1,1,1,
                                      1,0,0,0,0,0,0,0,0,1,
                                      1,0,0,0,0,1,0,0,0,1,
                                      1,0,0,0,1,1,0,0,0,1,
                                      1,0,0,0,0,1,0,0,0,1,
                                      1,0,0,0,0,1,0,0,0,1,
                                      1,0,0,0,0,1,0,0,0,1,
                                      1,0,0,0,1,1,1,0,0,1,
                                      1,0,0,0,0,0,0,0,0,1,
                                      1,1,1,1,1,1,1,1,1,1
```

```
                                                    };
          for(i=0;i<10;i++)   {
            for(j=0;j<10;j++)  {
                           if(C_1[i][j]==1) putpixel(x+j,y+i,3);
                           else putpixel(x+j,y+i,4);
            }
          }
return(tipo_1);

}
int cuad_2(int x, int y)  { int tipo_2=2;
                      int i,j;
                      int C_2[10][10]={1,1,1,1,1,1,1,1,1,1,
                                       1,0,0,0,0,0,0,0,0,1,
                                       1,0,0,1,1,1,1,0,0,1,
                                       1,0,0,1,0,0,1,0,0,1,
                                       1,0,0,1,0,0,1,0,0,1,
                                       1,0,0,0,0,1,0,0,0,1,
                                       1,0,0,0,1,0,0,0,0,1,
                                       1,0,0,1,1,1,1,1,0,1,
                                       1,0,0,0,0,0,0,0,0,1,
                                       1,1,1,1,1,1,1,1,1,1
                                              };
          for(i=0;i<10;i++)   {
            for(j=0;j<10;j++)  {
                           if(C_2[i][j]==1) putpixel(x+j,y+i,5);
                           else putpixel(x+j,y+i,2);

            }
          }
return(tipo_2);
}
```

4.3.3 Selección de compuertas

Al iniciar con los programas de prueba de la interfaz, nos encontramos con muchos supuestos y con muchos problemas por resolver, todo esto nos puede agobiar inmediatamente y no sabremos por donde empezar, así que como se mencionó en la metodología a usar se trabajara un problema a la vez. Más aún, debemos trabajar procurando tener todas las ventajas a nuestro favor, simulando y proponiendo condiciones que nos faciliten la solución al problema.

Para seleccionar una compuerta, primero debemos crearla (dibujarla) y almacenarla en la base (guardar atributos). Para lo anterior tomaremos ventaja, primero propondremos una estructura para guardar los datos:

```
struct comp {
            int tipo;
            int x,y;
            int id;
                  }gates[15];
```

Después crearemos las compuertas donde queramos y al mismo tiempo guardaremos los datos de cada una (tipo, color y posición).

```
for(i=0;i<7;i++){
            gates[i].id=i+1;
      gates[i].tipo=select(i,i*30,i*40);
      gates[i].x=i*30;
       gates[i].y=i*40;
}
```

Para seleccionar las compuertas nos desplazaremos por nuestra estructura, por medio de las flechas del teclado sumaremos o restaremos posiciones, presionando ESC señalaremos la compuerta que nos interesa y utilizaremos el algoritmo del programa pelota.c para moverla.

Las librerias gates.h y color.h seran sustituidas mas adelante por una sola que contendrá mas parámetros, que hay ambigüedad entre ellas.

```
// Armando Nava Linares
// Test para crear una base de datos
// Seleccion de compuertas por medio de las flechas del teclado

#include <graphics.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <c:\trabajos\armando\final\gates.h>
#include <c:\trabajos\armando\final\color.h>
#include <dos.h>
//sel4.cpp
int select(int gat,int X,int Y);
void codigo(int gat,int COLOR, int X, int Y);

struct comp {
            int tipo;
            int x,y;
            int id;
                  }gates[15];


int main(void)
{
      /* request auto detection */
      int gdriver = DETECT, gmode, errorcode;

      int i,num_max,compuert,temp,aux;
      char t,sel_g;
      int dis;
      void *sector_1,*sector_2,*sector_3,*sector_4;
      int equis, ye;

     /* initialize graphics and local variables */
     initgraph(&gdriver, &gmode, "");

     /* read result of initialization */
     errorcode = graphresult();
     if (errorcode != grOk)  /* an error occurred */
     {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1); /* terminate with an error code */
     }

      for(i=0;i<7;i++){
       gates[i].id=i+1;
      gates[i].tipo=select(i,i*30,i*40);
       gates[i].x=i*30;
        gates[i].y=i*40;
          }

outtextxy(200,100,"Ahora seleccione la compuerta,press ENTER");

// Numero de elementos de la base de datos

     for(i=0;i<100;i++){
       if(gates[i].id!=0)   num_max=gates[i].id;
```

```
       else break;
     }

// Corrimiento
if(gates[0].id!=0)  codigo(gates[0].tipo, RED,gates[0].x, gates[0].y);

  compuert=0;
  do
    {
    temp=compuert;
    t=inport(0x60);  /*entrada desde el teclado */
    //getch();
      switch(t)
             {
               case 72:  compuert--;
                       if( compuert==-1 ) {   compuert=num_max-1;
                                         temp=0;            }
                       aux=select(gates[temp].tipo, gates[temp].x,
gates[temp].y);
                       codigo(gates[compuert].tipo,
RED,gates[compuert].x, gates[compuert].y);
                       getch();
                        gotoxy(20,20);
                        printf("
");
                        gotoxy(20,20);
                        printf("Presione 2 veces ENTER para
seleccionar");
                        sel_g=inport(0x60);
                        getch();
                        if(sel_g==28) {
                                  dis=compuert;
                                  goto salida;
                                }
                       break;/*Cursor arriba */

               case 80:  compuert++;
                       if( compuert==num_max ) { compuert=0;
                                          temp=num_max-1;  }
                       aux=select(gates[temp].tipo, gates[temp].x,
gates[temp].y);
                       codigo(gates[compuert].tipo,
RED,gates[compuert].x, gates[compuert].y);
                       getch();
                        gotoxy(20,20);
                        printf("
");
                        gotoxy(20,20);
                        printf("Presione 2 veces ENTER para
seleccionar");
                        sel_g=inport(0x60);
                        getch();
                        if(sel_g==28) {
                                  dis=compuert;
```

```
                                    goto salida;
                                  }
                        break; /*Cursor abajo */

                default: break;

              }
  }while(t!=1); /*Mientras no se pulse esc */

salida:
      gotoxy(20,20);
      printf("                                                    ");
      gotoxy(20,20);
      printf("El discriminante es: %d,ya puede mover",dis);

/* Se colorea la compuerta elegida*/
codigo(gates[dis].tipo,YELLOW,gates[dis].x,gates[dis].y);

//Aqui se borran las lineas asociadas
//Se redibuja la imagen compuertas y lineas

do
  {
  t=inport(0x60);  /*entrada desde el teclado */
  switch(t)
      {

          case 72: /*Cursor arriba */

                gates[dis].y--;
                break;
          case 75:
                gates[dis].x--;
                break; /*Cursor izquierda*/
          case 77:
                gates[dis].x++;
                break; /*Cursor derecha */
          case 80:
                gates[dis].y++;
                break; /*Cursor abajo */
          default:break;

        }

   cleardevice();
             i=-1;
              do{   i++;
                  if(i!=dis)
codigo(gates[i].tipo,WHITE,gates[i].x,gates[i].y);

                }while(i<num_max);

            codigo(gates[dis].tipo,YELLOW,gates[dis].x,gates[dis].y);
            getch();
```

```
  }while(t!=1); /*Mientras no se pulse esc */

   aux=aux+1;
   /* clean up */
   getch();
   closegraph();
   return 0;
}

int select(int gat,int X,int Y)
 {
   switch(gat){
             case 0: nand(X,Y);
                   break;
             case 1: and(X,Y);
                   break;
             case 2: nor(X,Y);
                   break;
             case 3: or(X,Y);
                   break;
             case 4: not(X,Y);
                   break;
             case 5: exor(X,Y);
                   break;
             case 6: exnor(X,Y);
                   break;
        default: break;
}
return(gat);
}

void codigo(int gat,int COLOR,int X,int Y)
 {
   switch(gat){
             case 0: col_nand(COLOR,X,Y);
                   break;
             case 1: col_and(COLOR,X,Y);
                   break;
             case 2: col_nor(COLOR,X,Y);
                   break;
             case 3: col_or(COLOR,X,Y);
                   break;
             case 4: col_not(COLOR,X,Y);
                   break;
             case 5: col_exor(COLOR,X,Y);
                   break;
             case 6: col_exnor(COLOR,X,Y);
                   break;

        default: break;
   }
}
```

4.4 LECTURA DE COMANDOS

Para el uso de comandos, se utilizo un array, el cual contiene las cadenas de caracteres que utilizaremos como comandos validos. Mediante la función "*comando"* se compara una cadena de entrada con el conjunto de instrucciones validas, la función comando devuelve en que renglón se encuentra la instrucción.

```
char claves[10][20]={   "nuevo",              //Aquí se almacena el set
de comandos
                  "compuerta",
                   "union",
                   "resultado",
                   "salir"         };


int sl;     //variable  que  indica  si  la  cadena  pertenece  al  set  de
comandos
//Funcion comando
//Compara una cadena con el set de comandos. Devuelve un entero
correspondiente a la posicion de un comando
//Requiere como argumento una cadena
int comando(char comando[20])
{
  int i,j,comp;

  for(i=0;i<5;i++) {
            sl=1;
            j=-1;
                 do{
                   j=j++;

                   if(comando[j]==claves[i][j])  comp=1;
                   else if(comando[j]!=claves[i][j])  comp=0;
                   sl=sl*comp;

                     // if(sl==1) goto label00;


                   }while(comando[j]!=0);
                   if(sl==1)   goto label00;
              }

label00:
      return(i);
```

```
}
```

El código anterior se puede implementar de la siguiente, manera:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
//Programa claves.cpp
//Datos globales para la funcion comando
int sl;
int comando(char comando[20]);
char claves[5][20]={  "nuevo"    ,
                  "mover"    ,
                  "valores"  ,
                  "resultado",
                  "salir"
                            };


void main(void)
{
  int valor,temp;
  char pase[20];

do{

     do{
      printf("\n Teclee un comando: ");
      scanf("%s",&pase);

        valor=comando(pase);
       }while(sl!=1);


 switch(valor){
             case 0: printf("\n NUEVO");
                   temp=1;
                   break;
             case 1: printf("\n MOVER");
                   temp=1;
                   break;
             case 2: printf("\n VALORES");
                   temp=1;
                   break;
             case 3: printf("\n RESULTADO");
                   temp=1;
                   break;
             case 4: printf("\n SALIR");
                   temp=0;
                   break;
             default:break;
```

```
            }
}while(temp==1);

}
```

Debido a que solo se retorna el renglón en donde esta la instrucción, validaremos si la instrucción es valida por medio de la variable *sl* , solicitando instrucciones, hasta que se teclee alguna que sea valida.

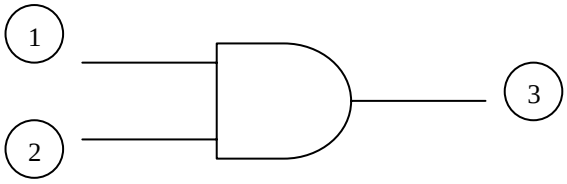
4.5 LA BASE DE DATOS

Podemos dibujar ahora compuertas en la pantalla, pero también requerimos manipularlas, por lo cual es necesario almacenar ciertos datos de la compuerta, como son posición, color, tipo, etc.

Componentes propuestos para la estructura compuerta.

```
struct compuerta{
   int x,y;         //Posicion de la compurta en la pantalla
   int tipo;        //Tipo de compuerta AND,OR,NOT,etc
   int id;          //Identificador de la compuerta,es el numero de compuerta
int nodo_id[3];     //Contiene el id de la compuerta conectada al nodo 1,2 y 3
int nodo_ter[3];    //Contiene la terminal a la que esta conectado el nodo1, 2
y 3
int solucion;       //contiene el status de la compuerta, si ya se resolvio
int log1,log2,log3; //Valores logicos de los nodos
}gate[100]   //Declaracion para el uso de 100 estructuras compuerta
             //Las coordenadas de las terminales  pueden ser derivadas a partir
             //  de la posicion de la compuerta
```

A continuación se muestra la designación de terminales de las compuertas



1 y 2 son entradas, 3 será la salida.

4.6 DESARROLLO DEL ENTORNO.

Para el inicio del entorno, vamos a hacer un recuento de los programas, funciones y librerías que nos servirán para iniciar la interfaz, es obvio que no se logró desarrollar todos los componentes en un solo paso, se desarrollaron varias versiones de prueba, solo se mencionaran las más importantes.

Programa	**Descripción**
Sel4.cpp	Manipulación de compuertas por medio de teclado.
Nand()	Grafica una compuerta nand
Gates.h	Gráfica las compuertas
Comando.h	Contiene el conjunto de comandos

Tabla 4.4 Programas y funciones a utilizar

4.6.1 Actualización de datos.

El programa sel4.cpp , la función nand() y la librería comando.h ya se ha visto con anterioridad. La librería gates.h es una extensión de la función nand, contiene es resto de las compuertas que se podrán graficar.

```
//Armando Nava Linares
//librería gates.h
// nand =1
// and  =2
// nor  =3
// or   =4
// not  =5
// exor =6
// exnor=7

void nand(int x,int y)
  {
   int i,j;
int
c_nand[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,
                0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                          };
  for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {
           if(c_nand[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);
          }
      }
}

void and(int x,int y)
  {
   int i,j;
   int
c_and[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
               1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,
               0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
               1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                          };
```

```
  for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {
           if(c_and[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);
          }
      }
    }

void nor(int x,int y)
  {
   int i,j;
   int
c_nor[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,1,1,1,1,1,
                  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                          };
  for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {
           if(c_nor[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);
          }
      }
  }
void or(int x,int y)
  {
     int i,j;
     int
c_or[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,
                  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
                                                                  };
  for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {
           if(c_or[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);
          }
      }
   }
void not(int x,int y)
  {
    int i,j;
    int
c_not[11][25]={0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,1,
                  0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                  };
  for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {
           if(c_not[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);
          }
      }
    }
void exnor(int x,int y)
  {
   int i,j;
   int
c_nor[11][25]={0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,1,1,1,1,1,1,
                  0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                  0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
                                                                  };
  for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {

            if(c_nor[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);

           }
        }
    }


void exor(int x,int y)
  {
     int i,j;
     int
c_or[11][25]={0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,
                  0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                  1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                  };

 for(j=0;j<25;j++)
       {
      for(i=0;i<11;i++)
          {
           if(c_or[i][j]==0)
             putpixel(x+j,y+i,0);
           else
             putpixel(x+j,y+i,15);
          }
      }
   }
```

La librería comando.h ha sido modificada para incluir nuevos comandos y se muestra a continuación:

```
//Armando Nava Linares
//Datos globales para la funcion comando
int sl;
char claves[13][20]={  "nuevo"    , //0
                       "mover"    , //1
                       "valores"  , //2
                       "resultado", //3
                       "salir"    , //4
                       "and"      , //5
                       "nand"     , //6
                       "or"       , //7
                       "nor"      , //8
                       "not"      , //9
                       "exor"     , //10
                       "exnor"      //11
                               };

//Funcion comando

//Compara una cadena con el set de comandos

//Devuelve un entero correspondiente a la posición de un comando

//Requiere como argumento una cadena

int comando(char comando[20])
{
  int i,j,comp;

  for(i=0;i<12;i++) {
            sl=1;
            j=-1;
                 do{
                   j=j++;

                   if(comando[j]==claves[i][j])  comp=1;
                   else if(comando[j]!=claves[i][j])  comp=0;
                   sl=sl*comp;

                     // if(sl==1) goto label00;


                   }while(comando[j]!=0);
                   if(sl==1)   goto label00;
              }

label00:
      return(i);


}
```

En esta librería (comando.h)se han adicionado las claves validas de nuestro programa, el orden en se encuentran puede variar, pero debe ser considerado cuando se efectúen cambios. Esta librería se debe de consultar para saber que comando son validos.

El programa claves.cpp puede ser considerado como un esqueleto de la interfaz. Se usa la función limpiar.h para dibujar lo que será el entono gráfico.

```
//Armando Nava Linares
//Funcion limpiar
void limpiar(void)
{

 int xmax = getmaxx();
 int ymax = getmaxy();

     line(0,0,xmax,0);

     line(0,0,0,ymax);

      line(0,ymax,xmax,ymax);

      line(xmax,ymax,xmax,0);

      line(30,0,30,ymax-100);

      line(0,ymax-70,xmax,ymax-70);

      line(0,ymax-100,xmax,ymax-100);

 //nand(3,10);
     outtextxy(30,420,"  nuevo   mover   valores   resultado   salir
");


 gotoxy(6,25);
    printf("
");
      gotoxy(6,25);

}
```

4.6.2 Versión 6

Por medio de la función limpiar mostramos los comandos disponibles y con la función comando, solicitamos la entrada de alguno de ellos. En el programa tesis6.cpp se muestran las pruebas para los comandos *valores*, *resultado* y *salir*. Dicha prueba esta basada en una compuerta nand, se solicitan los valores de entrada y visualiza el resultado. No se verifica si la entrada corresponde a un valor lógico valido.

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <c:\armando\tesis\comando.h>
#include <c:\armando\tesis\limpiar.h>
#include <c:\armando\tesis\nand.h>


//Los datos globales pueden ser consultados
//en COMANDO.H


void main(void)
{
  /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;

 //Constantes para el funcionamiento del programa
   int valor,temp;
   int x1,x2,resultado;
   char pase[20];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
```

```
      exit(1);
   }

   setcolor(9);



do{

     do{
      printf("\n Teclee un comando: ");
      scanf("%s",&pase);

        valor=comando(pase);
       }while(sl!=1);


 switch(valor){
             case 0: limpiar();
                   printf("Funcion NUEVO no disponible");
                   getch();
                   temp=1;
                   break;

             case 1: limpiar();
                   printf("Funcion MOVER no disponible");
                   getch();
                   temp=1;
                   break;

             case 2: limpiar(); //valores
                   nand(50,50);
                   printf("Cual es el valor de x1: ");
                   scanf("%d",&x1);
                   limpiar();
                   printf("Cual es el valor de x2: ");
                   scanf("%d",&x2);
                   temp=1;
                   break;

             case 3: limpiar(); //resultado
                   if(x1==1 && x2==1) resultado=0;
                   resultado=1;
                   printf("Con x1=%d y x2=%d, la salida es:
%d",x1,x2,resultado);
                   getch();
                   temp=1;
                   break;
             case 4: printf("\n SALIR");
                   temp=0;
                   break;
             default:break;

             }
```

```
}while(temp==1);

}
```

4.6.3 Versión 7

En el programa tesis7.cpp se muestra la adición del comando “nuevo”, el cual permite dibujar las compuertas en la pantalla, asignamos las coordenadas de la compuerta en incrementos de 60 para *x* y 30 para *y* , con el fin de ubicar las compuertas sin sobreponerlas.

De la misma manera en que la función comando() nos permite leer las instrucciones principales (nuevo, mover, valores, resultado, salir), podemos usarla para leer el tipo de compuerta (and, nand, or, nor, not, exor, exnor) a dibujar.

Cada vez que se cree una nueva compuerta, esta se gráfica, y después se guardan sus valores en la estructura

4.6.4 Versión 8

En el este programa se dispone ya de la función “mover”, la cual pudimos apreciar en el programa sel4, debido a requerimientos propios del programa, se ha

modificado parte del código del programa principal y las librerías graficas.h y comando.h. Esto con la finalidad de homologar el orden y la lógica de los valores de las compuertas.

Para el uso de la función mover se deben de crear o dibujar las compuertas a usar, mismas que deben ser dadas de alta en la estructura correspondiente, para guardar sus datos.

Los valores de las compuertas quedan como sigue:

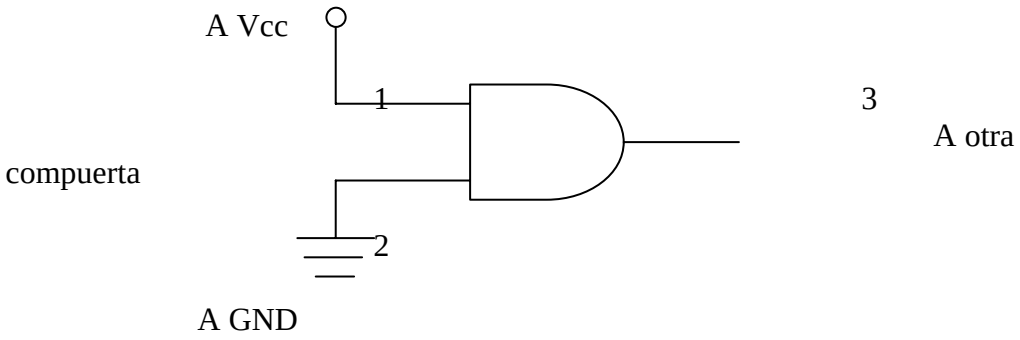
```
int nand =5;
int and  =6;
int nor  =7;
int or   =8;
int not  =9;
int exor =10;
int exnor=11;
```

4.6.5 Versión 10

Esta versión merece un trato especial ya que es donde se empiezan a hacer las asociaciones físicas y lógicas, los valores pueden ser tanto variables, como direcciones. El bloque que contiene la función mover nos permite seleccionar una compuerta, y saber cual de todas se eligió, esto lo sabemos mediante la variable “dis”, misma que nos proporciona todos los atributos de la compuerta seleccionada.

4.7 ASOCIACIÓN FÍSICA.

Dada una compuerta, sus entradas (1 y 2) pueden ser conectadas los siguientes valores Vcc, GND, o a la terminal de otra compuerta, este ultimo caso se ilustra con la terminal 3.



Para el caso de las entradas cuya asociación se llevara a cabo con Vcc o GND, se requiere de conocer los puntos (x, y) de dichas terminales, podría pensarse en incluir estas nuevas variables en la estructura de la compuerta, sin embargo, aprovecharemos que todas excepto **NOT** tienen las terminales 1, 2, 3 en los mismos puntos dentro del array, podemos derivar dichas coordenadas a partir de los puntos gates[].x y gates[].y, sumando o restando según sea el caso.

Valores derivados		
Terminal 1 (**ent1)**	Terminal 2 (**ent2**)	Terminal 3 (**salida**)
`Px1=gates[dis].x;` `Py1=gates[dis].y+3;`	`Px1=gates[dis].x;` `Py1=gates[dis].y+7;`	`Px1=gates[dis].x+25;` `Py1=gates[dis].y+5;`

Tabla 4.5 Valores derivados para las terminales de las compuertas

Para el caso del inversor (NOT) los valores derivados quedan como sigue:

Valores derivados del inversor		
Terminal 1 (**ent1)**	Terminal 2 (**ent2**)	Terminal 3 (**salida**)
`Px1=gates[dis].x;` `Py1=gates[dis].y+5;`	`Px1=gates[dis].x;` `Py1=gates[dis].y+5;`	`Px1=gates[dis].x+25;` `Py1=gates[dis].y+5;`

Tabla 4.5 Valores derivados para las terminales del inversor

Las variables **ent1** y **ent2** para el caso del inversor serán tomadas como una única terminal identificada por **ent1**.

Cuando la asociación física de las entradas (1 y 2) es con Vcc o GND, la rutina valores debe terminar colocando Vcc o GND a la entrada correspondiente. Nuevamente usamos un array para almacenar los símbolos de Vcc y GND.

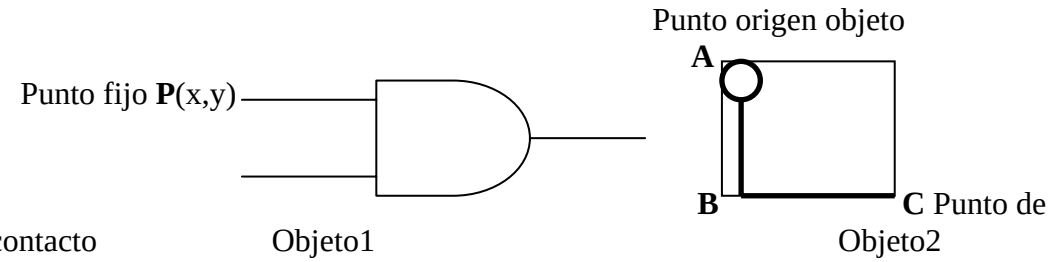
```
int    vcc[11][10]={0,1,1,1,0,0,0,0,0,0,
                    1,0,0,0,1,0,0,0,0,0,
                    1,0,0,0,1,0,0,0,0,0,
                    0,1,1,1,0,0,0,0,0,0,
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,1,0,0,0,0,0,0,0,
                    0,0,1,1,1,1,1,1,1,1, //Punto de contacto
                                        };

int    gnd[11][10]={0,0,0,0,1,1,1,1,1,1, //Punto de contacto
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    1,1,1,1,1,1,1,1,1,0,
                    0,0,0,0,0,0,0,0,0,0,
                    0,1,1,1,1,1,1,1,0,0,
                    0,0,0,0,0,0,0,0,0,0,
                    0,0,1,1,1,1,1,0,0,0,
                    0,0,0,0,0,0,0,0,0,0,
                                        };
```

Los unos en negritas indican los puntos de contacto, como son diferentes en ambos arrays, se debe nuevamente de derivar el punto donde será colocado Vcc o GND, más aún se debe de tomar en consideración a que terminal de entrada van (1, ó 2). Para el caso particular de Vcc en la entrada 1 y GND en la entrada 2, tenemos lo siguiente:

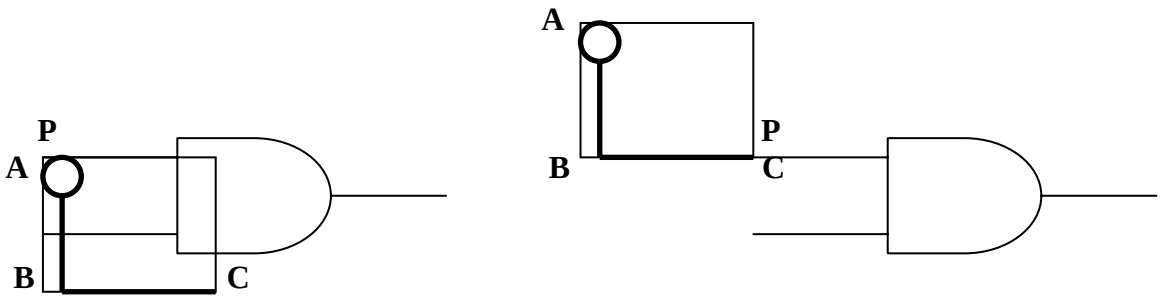
Vcc en la entrada 1 (Px1-10, Py1-10)

GND en la entrada 2 (Px1-10,Py1)

Px1 y Py1 son puntos derivados donde se localizan las terminales de entrada de la compuerta. La regla general para derivar nuestros nuevos puntos, es identificar hacia donde nos tenemos que desplazar para hacer coincidir nuestro punto de contacto con un punto fijo.



Consideremos como punto fijo P(x, y) del objeto 1.



Haciendo coincidir el punto de origen A del objeto 2, con el punto P, podemos ver que para hacer coincidir el punto de contacto C con el punto P, se tienen que hacer los desplazamientos AB hacia arriba y BC hacia la derecha.

Los mismos principios se aplican para los casos:

Vcc a entrada 1 Vcc a entrada 2	GND a entrada 1 GND a entrada 2	GND a entrada 1 Vcc a entrada 2

Ahora veremos que pasa cuando una terminal va a otra compuerta. En primer lugar tenemos que conservar la variable **dis**, salvar **Px1 en x_1** y **Py1** en **y_1**.

La variables **dis** nos indica que compuerta se selecciono en primer lugar, x_1 y y_1 nos indican las coordenadas de la terminal elegida.

Nuevamente hacemos uso del bloque seleccionar, esta vez guardando la variable dis como **dis2**, se selecciona la terminal deseada en la otra compuerta, cuyo punto se guarda en **x_2, y_2**, y por ultimo dibujamos una línea del punto (x_1, y_1) al (x_2, y_2);

4.8 ASOCIACIÓN LÓGICA.

Para la asociación lógica definiremos algunas nuevas variables, que formaran parte de la estructura compuerta, estas son, ent1, ent2, salida, mismas que declararemos de la siguiente forma:

```
int *ent1, *ent2;
int salida;
```

Como podemos observar, las entradas de la compuerta están definidas por los punteros *ent1* y *ent2*, y la terminal de salida, por la variable entera *salida*. El objetivo de esto es que varias entradas podrán apuntar a la dirección de una salida, Vcc o GND.

Faltan por definir los valores de las variable VCC, GND y ZETA, VCC indicara el estado lógico 1, GND el valor 0, y ZETA alta impedancia. Las asociaciones quedan de las siguiente manera:

```
gates[dis].ent1 = &VCC;                  //entrada1 a Vcc
gates[dis].ent1 = &GND;                  //entrada1 a GND
gates[dis].ent1 = &gates[dis2].salida;   //entrada1 a salida de otra
                                         //compuerta
gates[dis].ent1 =gates[dis2].ent1;       //entrada1 a entrada1 de otra
                                         //compuerta
gates[dis].ent1 =gates[dis2].ent2;       //entrada1 a entrada2 de otra
                                         //compuerta
```

4.9 SOLUCIÓN DEL DIAGRAMA LÓGICO

Comenzaremos por declarar una variable que nos indique si la compuerta ya se ha resuelto, la declaración queda como sigue:

```
int resuelta;
```

Esta variable tomara el valor de 0 si no se ha resuelto y de 1 si ya se resolvió.

Para efectos prácticos, inicializaremos las entradas de nuestras compuertas con el valor lógico 1, con el fin de que las entradas tenga un valor lógico, en caso de que se omita el uso de alguna de sus entradas. Esto en analogía con las compuertas TTL, que toman la lata impedancia como un 1 lógico. La inicialización se hace al momento de que se crea una nueva compuerta.

```
gates[ID].ent1=&ZETA;
gates[ID].ent2=&ZETA;
```

Cuando asociamos la salida de una compuerta sin resolver, a la entrada de otra compuerta, tenemos como resultado un valor incierto, y es por esto que requerimos de una función que nos indique si hay valores lógicos a la entrada de una compuerta y en caso de haberlo comenzar a resolver la compuerta.

```
int booleano(int a){

int status;
      if(a==0 || a==1) status=1;
      else status=0;

return(status);
}
```

La función booelano nos devuelve un 1 si el parámetro que le pasamos es un valor lógico valido 1 ó 0, de no ser así devuelve un 0.

Para resolver cada tipo de compuerta contamos ya con funciones propias del lenguaje C, que manejan bits, como son AND, OR y OR exclusiva, aunque contamos con el operador ! , implementaremos un inversor.

```
int f_not(int a){   //Funcion Negacion

      int res;

     if(a==1) res=0;
     if(a==0) res=1;

return(res);
}
```

Con este inversor podemos implementar las demás funciones lógicas NAND, NOR y EXNOR. Antes de comenzar a resolver las funciones lógicas de cada compuerta debemos asegurarnos de que por lo menos exista una compuerta declarada, para lo cual adicionaremos otra variable, que será inicializada a 1, en cuanto se cree una nueva compuerta.

```
gates[ID].en_uso=1;
```

Mediante un ciclo for analizaremos cada compuerta, en caso de que estén presentes valores lógicos, se efectuara la función correspondiente a la compuerta y se almacenara el resultado en la variable salida, quedando el bloque que da solución al diagrama como sigue:

```
num_max = ID;

if(gates[0].en_uso==1) {


    do {
            for(i=0; i<num_max; i++){
            //    TIPO=gate[i].tipo;
            //    A= *gate[i].ent1;
          //      B= *gate[i].ent2;

                  if ( booleano(*gates[i].ent1) &&
booleano(*gates[i].ent2) ){

                        switch(gates[i].tipo){

                        case 5: temp_s=f_and( *gates[i].ent1,
*gates[i].ent2 );
                              temp_s=f_not(temp_s);
                              gates[i].salida= temp_s;
                              gates[i].resuelta=1;
                              break;

                        case 6: temp_s=f_and( *gates[i].ent1,
*gates[i].ent2 );
                              gates[i].salida= temp_s;
                              gates[i].resuelta=1;
                              break;

                        case 7: temp_s=f_or( *gates[i].ent1,
*gates[i].ent2 );
                              temp_s=f_not(temp_s);
                              gates[i].salida= temp_s;
                              gates[i].resuelta=1;
                              break;

                        case 8: temp_s=f_or( *gates[i].ent1,
*gates[i].ent2 );
                              gates[i].salida= temp_s;
                              gates[i].resuelta=1;
                              break;

                        case 9: temp_s=f_not( *gates[i].ent1);
                              gates[i].salida= temp_s;
```

```
                              gates[i].resuelta=1;
                              break;

                        case 10:temp_s=f_xor( *gates[i].ent1,
*gates[i].ent2 );
                              gates[i].salida= temp_s;
                              gates[i].resuelta=1;
                              break;

                        case 11:temp_s=f_xor( *gates[i].ent1,
*gates[i].ent2 );
                              temp_s=f_not(temp_s);
                              gates[i].salida= temp_s;
                              gates[i].resuelta=1;
                              break;

                        default: break;

                        }

                  }

            }
temp_r=1;
for(i=0; i<num_max; i++) temp_r=(temp_r)*(gates[i].resuelta);


    }while(temp_r==0);

      for(i=0; i<num_max; i++){
      if(gates[i].salida==1)  graficar(gates[i].x, gates[i].y,
gates[i].tipo, RED);
      if(gates[i].salida==0)  graficar(gates[i].x, gates[i].y,
gates[i].tipo, LIGHTBLUE);
      }


}


for(i=0; i<num_max; i++) gates[i].resuelta=0;
```

Para indicar el estado de la salida de cada compuerta, usaremos los colores ROJO para ALTO y AZUL para BAJO, de esta manera identificaremos rápidamente el resultado de nuestras compuertas.

CONCLUSIONES.

CONCLUSIONES.

Podemos señalar que, aunque este simulador tiene muchas características que pulir y agregar, fuera de hacerlo menos, lo hace más interesante, ya que tenemos código fuente de donde partir para agregar las características que deseemos incorporar. Si de antemano hubiéramos buscado incorporar cada característica deseada al simulador, nos habríamos enfrentado con muchos problemas, mismos que a la larga se convertirían en obstáculos para seguir adelante.

Se experimentaron complicaciones cuando se quiso hacer una interfaz basada en el ratón, como se partía desde cero, hubiera sido más complicado terminarla, así que se opto por la interfaz de comandos, misma que a la larga nos proporcionó muchas funciones útiles, más aún, el algoritmo utilizado para resolver el diagrama lógico puede ser usado en un futuro por la interfaz basada en el ratón. Sin embargo esta experiencia fue provechosa, porque su construyeron nuevas funciones que involucran el uso del ratón.

Si bien hay que estar actualizado día con día, también es importante contar con las bases suficientes que nos permitan lograr nuestros objetivos . Este simulador se desarrollo con software, que es considerado viejo, pero que por ser de uso general nos proporciona las bases para estudiar otros lenguajes de programación más nuevos. Cabe mencionar que no se utilizo el lenguaje C a todo su potencial, eso con la finalidad de que el programa fuera lo más sencillo de posible y fácil de entender.

El implementar un simulador gráfico como este, trajo consigo muchas versiones del programa y otros programas utilitarios, algunos no se utilizaron y otros fueron fundamentales para poder terminar este trabajo. Esta diversidad de programas trajo consigo conocimientos nuevos, de manera que fue fácil perderse en el camino, de aquí la importancia de contar con una metodología que nos permitiese llegar a alcanzar nuestros objetivos.

Contamos ahora con código fuente, y se abren las posibilidades de enriquecer las funciones con las que cuenta este simulador, tanto de nuestra parte como por parte de otros programadores, ya que un esfuerzo colectivo trae muchas más ideas.

Queda comprobado una vez más el carácter general del lenguaje C, mismo que nos permite programar desde bases de datos, sistemas operativos, editores de texto, y en nuestro caso un simulador de compuertas digitales.

ANEXOS.

ANEXO 1

Listado del programa

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <c:\ver9\comando.h>
#include <c:\ver9\limpiar.h>
#include <c:\ver9\graficas.h>
#include <c:\ver9\terminal.h>
#include <c:\ver9\funcion.h>

//Armando Nava Linares
//Comando mover disponible



int ID;
struct compuerta {
             int tipo;  //Tipo de compuerta
             int x,y;   //Posicion
             int id;    //Identificador dentro de la estructura
   int *ent1; //puntero para el valor de entrada1
   int *ent2; //puntero para el valor de entrada2
   int salida;    //variable que almacena el valor de la salida 3
   int en_uso;    //variable que indica si la compuerta esta en uso
   int resuelta;  //indica si ya se resolvio la compuerta
       }gates[100];




//int select(int gat,int X,int Y);
//void codigo(int gat,int COLOR, int X, int Y);




//Los datos globales pueden ser consultados
//en COMANDO.H


void main(void)
{
  /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;
   int i, num_max;
 //Variables para el funcionamiento del programa
   int valor,temp;
   int x1,x2,resultado, compuerta;
   char pase[20],gate[20];
   int x_pos=30,y_pos=30;


   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "\\tc\\bgi");

/* Variables para mover*/
```

```
char t;
int compuert;
int dis, dis2;

/**********************/

/*Variables para valores*/
int terminal;
//enum {GND,Vcc, g};

int GND=0;
int VCC=1;
int ZETA=1;

char logo[20];
int T;

int Px1, Py1;
/************************/

/***Variables para unir***/
int x_1, x_2, y_1, y_2;

/******************/

int temp_s;
int temp_r;

int TIPO, A,B;



   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }

   xmax=getmaxx();
   ymax=getmaxy();

   setcolor(9);


ID=0; //Identificador del la compuerta

do{

     do{
        limpiar();
           printf("\ Teclee un comando: ");
             scanf("%s",&pase);   //Cadena a comparar
       valor=comando(pase);      //Resultado de la comparacion

       }while(sl!=1); //Mientras no exista coincidencia con algun comando


       switch(valor){
              case 0: limpiar();  //NUEVO

                      if(y_pos>ymax)  {      printf("Se  excedio  el  numero  maximo  de
copuertas");
                                       break;
```

```
                }

if(x_pos>(xmax-60))  { x_pos=60;
                       y_pos=y_pos+30;
                     }
else if(x_pos<(xmax-60))   x_pos=x_pos+30;



printf("Tipo de compuerta: ");
scanf("%s",&gate);

     compuerta=comando(gate);

         switch(compuerta){

                case 5: graficar(x_pos,y_pos,compuerta,WHITE );
                        //Almacena datos NAND
                        gates[ID].id=ID;
                        gates[ID].tipo=compuerta;
                        gates[ID].x=x_pos;
                        gates[ID].y=y_pos;
                        gates[ID].en_uso=1;
                        gates[ID].ent1=&ZETA;
                        gates[ID].ent2=&ZETA;
                        ID++;
                        break;

                case 6: graficar(x_pos,y_pos, compuerta,WHITE);
                        //Almacena datos AND
                        gates[ID].id=ID;
                        gates[ID].tipo=compuerta;
                        gates[ID].x=x_pos;
                        gates[ID].y=y_pos;
                        gates[ID].en_uso=1;
                        gates[ID].ent1=&ZETA;
                        gates[ID].ent2=&ZETA;
                        ID++;
                        break;
                case 7: graficar(x_pos,y_pos, compuerta,WHITE);
                        //Almacena datos NOR
                        gates[ID].id=ID;
                        gates[ID].tipo=compuerta;
                        gates[ID].x=x_pos;
                        gates[ID].y=y_pos;
                        gates[ID].en_uso=1;
                        gates[ID].ent1=&ZETA;
                        gates[ID].ent2=&ZETA;
                        ID++;
                        break;
                case 8: graficar(x_pos,y_pos, compuerta,WHITE);
                        //Almacena datos OR
                        gates[ID].id=ID;
                        gates[ID].tipo=compuerta;
                        gates[ID].x=x_pos;
                        gates[ID].y=y_pos;
                        gates[ID].en_uso=1;
                        gates[ID].ent1=&ZETA;
                        gates[ID].ent2=&ZETA;
                        ID++;
                        break;
                case 9: graficar(x_pos,y_pos, compuerta,WHITE);
                        //Almacena datos NOT
                        gates[ID].id=ID;
                        gates[ID].tipo=compuerta;
                        gates[ID].x=x_pos;
                        gates[ID].y=y_pos;
                        gates[ID].en_uso=1;
```

```
                                        gates[ID].ent1=&ZETA;
                                        gates[ID].ent2=&ZETA;
                                        ID++;
                                        break;
                                case 10:graficar(x_pos,y_pos, compuerta,WHITE);
                                        //Almacena datos EXOR
                                        gates[ID].id=ID;
                                        gates[ID].tipo=compuerta;
                                        gates[ID].x=x_pos;
                                        gates[ID].y=y_pos;
                                        gates[ID].en_uso=1;
                                        gates[ID].ent1=&ZETA;
                                        gates[ID].ent2=&ZETA;
                                        ID++;
                                        break;
                                case 11:graficar(x_pos,y_pos, compuerta,WHITE);
                                        //Almacena datos EXNOR
                                        gates[ID].id=ID;
                                        gates[ID].tipo=compuerta;
                                        gates[ID].x=x_pos;
                                        gates[ID].y=y_pos;
                                        gates[ID].en_uso=1;
                                        gates[ID].ent1=&ZETA;
                                        gates[ID].ent2=&ZETA;
                                        ID++;
                                        break;
                                default: break;
                                   }
                        getch();
                        temp=1;
                        break;

               case 1: limpiar();   //MOVER
                        printf("Use las flechas");
/*******************************************/
    num_max=ID;
    compuert=0;

do
    {
    temp=compuert;
    t=inport(0x60);  /*entrada desde el teclado */
    //getch();
       switch(t)
                {
                  case 72:  compuert--;
                            if( compuert==-1 ) {   compuert=num_max-1;
                                                   temp=0;            }
graficar(gates[temp].x, gates[temp].y,gates[temp].tipo,WHITE);
graficar(gates[compuert].x, gates[compuert].y,gates[compuert].tipo,RED);
//aux=select(gates[temp].tipo, gates[temp].x, gates[temp].y);
//codigo(gates[compuert].tipo, RED,gates[compuert].x, gates[compuert].y);
                            getch();
                              gotoxy(20,20);
                              printf("                                                  ");
                              gotoxy(20,20);
                              printf("Presione 2 veces ENTER para seleccionar");
                              t =inport(0x60);
                              if (t==28) {
                              dis= compuert;
                              goto salida; }
                              getch();
                              //if(sel_g==28) {
                              //          dis=compuert;
                              //          goto salida;
                              //        }
                            break;/*Cursor arriba */
```

```
                case 80:  compuert++;
                          if( compuert==num_max ) { compuert=0;
                                                    temp=num_max-1;  }

graficar(gates[temp].x, gates[temp].y,gates[temp].tipo,WHITE);
graficar(gates[compuert].x, gates[compuert].y,gates[compuert].tipo,RED);
//aux=select(gates[temp].tipo, gates[temp].x, gates[temp].y);
//codigo(gates[compuert].tipo, RED,gates[compuert].x, gates[compuert].y);
                          getch();
                            gotoxy(20,20);
                            printf("                                                    ");
                            gotoxy(20,20);
                            printf("Presione 2 veces ENTER para seleccionar");
                            t =inport(0x60);
                            if (t==28) {dis=compuert;
                            goto salida;}
                            getch();
                            //if(sel_g==28) {
                            //          dis=compuert;
                            //          goto salida;
                            //        }
                          break; /*Cursor abajo */

                //case 28: dis= compuert;
                //    goto salida;


                 default: break;

               }
  }while(t!=1); /*Mientras no se pulse esc */


salida:
       gotoxy(20,20);
       printf("                                                  ");
       gotoxy(20,20);
       printf("El discriminante es: %d,ya puede mover",dis);

/* Se colorea la compuerta elegida*/

graficar(gates[dis].x, gates[dis].y,gates[dis].tipo,YELLOW);
//codigo(gates[dis].tipo,YELLOW,gates[dis].x,gates[dis].y);

//Aqui se borran las lineas asociadas
//Se redibuja la imagen compuertas y lineas



do
  {
  t=inport(0x60);  /*entrada desde el teclado */
  switch(t)
      {

           case 72: /*Cursor arriba */

                  gates[dis].y--;
                  break;
           case 75:
                  gates[dis].x--;
                  break; /*Cursor izquierda*/
           case 77:
                  gates[dis].x++;
                  break; /*Cursor derecha */
           case 80:
                  gates[dis].y++;
                  break; /*Cursor abajo */
```

```
           default:break;

         }

   cleardevice();
               i=-1;
                do{   i++;
                     if(i!=dis)
        graficar(gates[i].x, gates[i].y,gates[i].tipo,WHITE);
        //codigo(gates[i].tipo,WHITE,gates[i].x,gates[i].y);

                  }while(i<num_max);

        graficar(gates[dis].x, gates[dis].y,gates[dis].tipo,YELLOW);
        //codigo(gates[dis].tipo,YELLOW,gates[dis].x,gates[dis].y);
              t =inport(0x60);
              getch();

  }while(t!=1); /*Mientras no se pulse esc */

graficar(gates[dis].x, gates[dis].y,gates[dis].tipo,WHITE);


/*******************************/



                    // printf("Funcion MOVER no disponible");
                    // getch();
                     temp=1;
                     break;

        case 2: limpiar(); //VALORES
   /************PRIMERO SE SELECCIONA LA COMPUERTA************/
   num_max=ID;
   compuert=0;
   do
    {
        temp=compuert;
        t=inport(0x60);  /*entrada desde el teclado */
        //getch();
        switch(t)
                {
                  case 72:  compuert--;
                            if( compuert==-1 ) {   compuert=num_max-1;
                                                   temp=0;            }
        graficar(gates[temp].x, gates[temp].y,gates[temp].tipo,WHITE);
        graficar(gates[compuert].x, gates[compuert].y,gates[compuert].tipo,RED);
                             getch();
                             gotoxy(20,20);
                             printf("                                                ");
                             gotoxy(20,20);
                             printf("Presione 2 veces ENTER para seleccionar");
                             t =inport(0x60);
                             if (t==28) {
                             dis= compuert;
                             goto salida2; }
                             getch();
                             //if(sel_g==28) {
                             //          dis=compuert;
                             //          goto salida;
                             //        }
                            break;/*Cursor arriba */

                  case 80:  compuert++;
                            if( compuert==num_max ) { compuert=0;
                                                      temp=num_max-1;  }
```

```
 graficar(gates[temp].x, gates[temp].y,gates[temp].tipo,WHITE);
 graficar(gates[compuert].x, gates[compuert].y,gates[compuert].tipo,RED);
                       getch();
                       gotoxy(20,20);
                       printf("                                                ");
                       gotoxy(20,20);
                       printf("Presione 2 veces ENTER para seleccionar");
                       t =inport(0x60);
                       if (t==28) {dis=compuert;
                       goto salida2;}
                       getch();
                       //if(sel_g==28) {
                       //          dis=compuert;
                       //          goto salida;
                       //        }
                     break; /*Cursor abajo */

           //case 28: dis= compuert;
           //    goto salida;


            default: break;

         }
 }while(t!=1); /*Mientras no se pulse esc */
 salida2:

 /* Se colorea la compuerta elegida*/
 graficar(gates[dis].x, gates[dis].y,gates[dis].tipo,YELLOW);
 /***************TERMINA SELECCION PARA VALORES*************/
limpiar();
printf("Terminal (1, 2, 3): ");
scanf("%d", &terminal);
 switch(terminal) {

 case 1: //Entrada1
        limpiar();
        //Almacenar coordenada (x_a,y_a);

        Px1=gates[dis].x; //Vcc en 1
        Py1=gates[dis].y+3;

                       if(gates[dis].tipo == 9){    //Para el caso de NOT
                       Px1=gates[dis].x; //Vcc en 1
                       Py1=gates[dis].y+5;
                       }

        printf("A (Vcc, GND, gate): ");  //Definir los valores de Vcc, GND, g
        scanf("%s",&logo);
        T = comando(logo);
               //limpiar();
               //printf("commando: %d",T);
               //getch();
               //Switch (logo){

         if(T==12){  //Vcc en entrada 1
         term(Px1-10, Py1-10,1,4);
         gates[dis].ent1 = &VCC;
         //line (Px1, Py1, Px1-10, Py1);
         }

         if(T==13){  //GND en entrada 1
         term(Px1-20,Py1,4,4);
         gates[dis].ent1 = &GND;
         }

         if(T==14){  //1 va a otra terminal
```

```
                        //Seleccionar otra compuerta
                        //Guardar datos anteriores
                        x_1 = Px1;
                        y_1 = Py1;
                 /*********Seleccion de otra compuerta*********/
                       limpiar();
                       printf("Seleccione otra compuerta: ");

               compuert=0;
               do
               {
                             temp=compuert;
                             t=inport(0x60);  /*entrada desde el teclado */
                             //getch();
                             switch(t)
                             {
                             case 72:  compuert--;
                             if( compuert==-1 ) {   compuert=num_max-1;
                                                temp=0;            }
                             graficar(gates[temp].x,
gates[temp].y,gates[temp].tipo,WHITE);
                             graficar(gates[compuert].x,
gates[compuert].y,gates[compuert].tipo,RED);
                             getch();
                             gotoxy(20,20);
                             printf("                                                  ");
                             gotoxy(20,20);
                             printf("Presione 2 veces ENTER para seleccionar");
                             t =inport(0x60);
                             if (t==28) {
                             dis2= compuert;
                             goto salida3; }
                             getch();
                             //if(sel_g==28) {
                             //          dis=compuert;
                             //          goto salida;
                             //        }
                             break;/*Cursor arriba */

                             case 80:  compuert++;
                             if( compuert==num_max ) { compuert=0;
                                                temp=num_max-1;  }

                             graficar(gates[temp].x,
gates[temp].y,gates[temp].tipo,WHITE);
                             graficar(gates[compuert].x,
gates[compuert].y,gates[compuert].tipo,RED);
                             getch();
                             gotoxy(20,20);
                             printf("                                                  ");
                             gotoxy(20,20);
                             printf("Presione 2 veces ENTER para seleccionar");
                             t =inport(0x60);
                             if (t==28) {dis2=compuert;
                             goto salida3;}
                             getch();
                             //if(sel_g==28) {
                             //          dis=compuert;
                             //          goto salida;
                             //        }
                             break; /*Cursor abajo */

                             //case 28: dis= compuert;
                             //    goto salida;


                             default: break;
```

```
                }
        }while(t!=1); /*Mientras no se pulse esc */


salida3:
/* Se colorea la compuerta elegida*/
graficar(gates[dis2].x, gates[dis2].y,gates[dis2].tipo,YELLOW);

limpiar();
printf("Terminal (1, 2, 3): ");
scanf("%d", &terminal);
switch(terminal) {
        case 1: //a Entrada 1 a Entrada 1
                limpiar();
                //Almacenar coordenada (x_a,y_a);
                x_2=gates[dis2].x; //Vcc en 1
                y_2=gates[dis2].y+3;
                gates[dis].ent1=gates[dis2].ent1;
                break;

        case 2: //a Entrada 1 a Entrada 2
                x_2=gates[dis2].x;
                y_2=gates[dis2].y+7;
                gates[dis].ent1=gates[dis2].ent2;
                break;

        case 3: //a Entrada 1 a SALIDA 3
                x_2= gates[dis2].x+25;
                y_2= gates[dis2].y+5;
                gates[dis].ent1=&gates[dis2].salida;
                break;
                  }
 setcolor(RED);
 line(x_1, y_1, x_2, y_2);

 }  //if 14 terminal 1
/************TERMINAL 1 COMPLETA***************/


                        //case 12: //Vcc colocar punto (x,y) a Vcc
                        //case 13: //GND colocar punto (x,y) a GND
                        //case 14: //Seleccionar un punto en otra compuerta
                        //Pendiente la union logica
                        break;

case 2: //Entrada2
                limpiar();
                //Almacenar coordenada (x_a,y_a);
                Px1=gates[dis].x; //Vcc en 1
                Py1=gates[dis].y+7;

                if(gates[dis].tipo == 9){    //Para el caso de NOT
                Px1=gates[dis].x; //Vcc en 1
                Py1=gates[dis].y+5;
                }


                printf("A (Vcc, GND, gate): ");
                //Definir los valores de Vcc, GND, g
                scanf("%s",&logo);
                T = comando(logo);
                //limpiar();
                //printf("commando: %d",T);
                //getch();
                        //Switch (logo){


 if(T==12){  //Vcc en 2
```

```
                          //term(Px1-20, Py1-10,2,4);
                          //gates[dis].ent2=&VCC;

                                 if(gates[dis].tipo==9){
                                    term(Px1-10, Py1-10,1,4);
                                    gates[dis].ent1 = &VCC;
                                 }

                                 else{
                                   term(Px1-20, Py1-10,2,4);
                                   gates[dis].ent2=&VCC;
                                 }

                          }

                 if(T==13){  //GND en 2

                          //term(Px1-10,Py1,3,4);
                          //gates[dis].ent2=&GND;

                                 if(gates[dis].tipo==9){
                                    term(Px1-10, Py1,3,4);
                                    gates[dis].ent1 = &GND;
                                 }

                                 else{
                                    term(Px1-10,Py1,3,4);
                                    gates[dis].ent2=&GND;
                                 }

                          }

/*************COPIA TERMINAL1*****************/
                 if(T==14){  //terminal 2 va a otra terminal
                           //Seleccionar otra compuerta
                           //Guardar datos anteriores
                           x_1 = Px1;
                           y_1 = Py1;
/***************Seleccion de otra compuerta*****************/
               limpiar();
               printf("Seleccione otra compuerta: ");

               compuert=0;
               do
               {
                            temp=compuert;
                            t=inport(0x60);  /*entrada desde el teclado */
                            //getch();
                            switch(t)
                            {
                            case 72:  compuert--;
                            if( compuert==-1 ) {   compuert=num_max-1;
                                                   temp=0;            }
                            graficar(gates[temp].x,
gates[temp].y,gates[temp].tipo,WHITE);
                            graficar(gates[compuert].x,
gates[compuert].y,gates[compuert].tipo,RED);
                            getch();
                            gotoxy(20,20);
                            printf("                                                    ");
                            gotoxy(20,20);
                            printf("Presione 2 veces ENTER para seleccionar");
                            t =inport(0x60);
                            if (t==28) {
                            dis2= compuert;
                            goto salida4; }
                            getch();
```

```
                            //if(sel_g==28) {
                            //          dis=compuert;
                            //          goto salida;
                            //        }
                            break;/*Cursor arriba */

                            case 80:  compuert++;
                            if( compuert==num_max ) { compuert=0;
                                                temp=num_max-1;  }

                            graficar(gates[temp].x,
gates[temp].y,gates[temp].tipo,WHITE);
                            graficar(gates[compuert].x,
gates[compuert].y,gates[compuert].tipo,RED);
                            getch();
                            gotoxy(20,20);
                            printf("                                                    ");
                            gotoxy(20,20);
                            printf("Presione 2 veces ENTER para seleccionar");
                            t =inport(0x60);
                            if (t==28) {dis2=compuert;
                            goto salida4;}
                            getch();
                            //if(sel_g==28) {
                            //          dis=compuert;
                            //          goto salida;
                            //        }
                            break; /*Cursor abajo */

                            //case 28: dis= compuert;
                            //    goto salida;


                            default: break;

                            }
                      }while(t!=1); /*Mientras no se pulse esc */


              salida4:
              /* Se colorea la compuerta elegida*/
              graficar(gates[dis2].x, gates[dis2].y,gates[dis2].tipo,YELLOW);

                      limpiar();
       printf("Terminal (1, 2, 3): ");   //terminal de la otra compuerta
       scanf("%d", &terminal);
              switch(terminal) {
                      case 1: //a Entrada 2 a Entrada1
                              limpiar();
                              //Almacenar coordenada (x_a,y_a);
                              x_2=gates[dis2].x; //Vcc en 1
                              y_2=gates[dis2].y+3;
                              gates[dis].ent2=gates[dis2].ent1;
                              break;

                      case 2: //a Entrada 2 a entrada 2
                              x_2=gates[dis2].x; //Vcc en 1
                              y_2=gates[dis2].y+7;
                                gates[dis].ent2=gates[dis2].ent2;
                              break;

                      case 3: //a Entrada 2 a Salida 3
                              x_2= gates[dis2].x+25;
                              y_2= gates[dis2].y+5;
                                gates[dis].ent2=&gates[dis2].salida;
                              break;
                                }
               setcolor(RED);
```

```
                line(x_1, y_1, x_2, y_2);

                }  //if 14 terminal 1

/*************FIN COPIA*********************/


                                        //case 12: //Vcc colocar punto (x,y) a Vcc
                                        //case 13: //GND colocar punto (x,y) a GND
                                        //case 14: //Seleccionar un punto en otra compuerta
                                        //Pendiente la union logica
                                        break;


               case 3: //Salida 3 a alguna terminal
                               //Salvar Id compuerta anterior
                               //Seleccionar nueva compuerta ID2
                               //Seleccionar terminal ID2
                               x_1=gates[dis].x+25; //Salvar terminad 3 ID1
                               y_1=gates[dis].y+5;
                               //Seleccionar compuerta nueva
                        limpiar();
                        printf("Seleccione otra compuerta: ");

                        compuert=0;
                        do
                        {
                               temp=compuert;
                               t=inport(0x60);  /*entrada desde el teclado */
                               //getch();
                               switch(t)
                               {
                               case 72:  compuert--;
                               if( compuert==-1 ) {   compuert=num_max-1;
                                                      temp=0;            }
                               graficar(gates[temp].x,
gates[temp].y,gates[temp].tipo,WHITE);
                               graficar(gates[compuert].x,
gates[compuert].y,gates[compuert].tipo,RED);
                               getch();
                               gotoxy(20,20);
                               printf("                                                  ");
                               gotoxy(20,20);
                               printf("Presione 2 veces ENTER para seleccionar");
                               t =inport(0x60);
                               if (t==28) {
                               dis2= compuert;
                               goto salida5; }
                               getch();
                               //if(sel_g==28) {
                               //          dis=compuert;
                               //          goto salida;
                               //        }
                               break;/*Cursor arriba */

                               case 80:  compuert++;
                               if( compuert==num_max ) { compuert=0;
                                                    temp=num_max-1;  }

                               graficar(gates[temp].x,
gates[temp].y,gates[temp].tipo,WHITE);
                               graficar(gates[compuert].x,
gates[compuert].y,gates[compuert].tipo,RED);
                               getch();
                               gotoxy(20,20);
                               printf("                                                  ");
```

```
                        gotoxy(20,20);
                        printf("Presione 2 veces ENTER para seleccionar");
                        t =inport(0x60);
                        if (t==28) {dis2=compuert;
                        goto salida5;}
                        getch();
                        //if(sel_g==28) {
                        //          dis=compuert;
                        //          goto salida;
                        //        }
                        break; /*Cursor abajo */

                        //case 28: dis= compuert;
                        //    goto salida;


                        default: break;

                        }
                }while(t!=1); /*Mientras no se pulse esc */


                salida5:
                /* Se colorea la compuerta elegida*/
                graficar(gates[dis2].x, gates[dis2].y,gates[dis2].tipo,YELLOW);

        limpiar();
        printf("Terminal (1, 2): ");   //terminal de la otra compuerta
        scanf("%d", &terminal);
        switch(terminal) {
                case 1: //Salida 3 a Entrada1
                        limpiar();
                        x_2=gates[dis2].x;
                        y_2=gates[dis2].y+3;
                        gates[dis2].ent1=&gates[dis].salida;
                        break;

                case 2: //Salida 3 a entrada 2
                        x_2=gates[dis2].x;
                        y_2=gates[dis2].y+7;
                        gates[dis2].ent2=&gates[dis].salida;
                        break;

                default: break;
                }

                setcolor(RED);
                line(x_1, y_1, x_2, y_2);



        default:break;  //Valor no valido

        }

        temp=1;
        break;


/*****************************************************/

       case 3: limpiar(); //resultado
                num_max = ID;

if(gates[0].en_uso==1) {
```

```
    do {
               for(i=0; i<num_max; i++){
              //       TIPO=gate[i].tipo;
              //       A= *gate[i].ent1;
             // B= *gate[i].ent2;

                        if ( booleano(*gates[i].ent1) && booleano(*gates[i].ent2) ){

                                switch(gates[i].tipo){

                                case 5: temp_s=f_and( *gates[i].ent1, *gates[i].ent2 );
                                       temp_s=f_not(temp_s);
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                case 6: temp_s=f_and( *gates[i].ent1, *gates[i].ent2 );
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                case 7: temp_s=f_or( *gates[i].ent1, *gates[i].ent2 );
                                       temp_s=f_not(temp_s);
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                case 8: temp_s=f_or( *gates[i].ent1, *gates[i].ent2 );
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                case 9: temp_s=f_not( *gates[i].ent1);
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                case 10:temp_s=f_xor( *gates[i].ent1, *gates[i].ent2 );
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                case 11:temp_s=f_xor( *gates[i].ent1, *gates[i].ent2 );
                                       temp_s=f_not(temp_s);
                                       gates[i].salida= temp_s;
                                       gates[i].resuelta=1;
                                       break;

                                default: break;

                                }

                        }

               }
temp_r=1;
for(i=0; i<num_max; i++) temp_r=(temp_r)*(gates[i].resuelta);


    }while(temp_r==0);

       for(i=0; i<num_max; i++){
       if(gates[i].salida==1)  graficar(gates[i].x, gates[i].y, gates[i].tipo, RED);
       if(gates[i].salida==0)       graficar(gates[i].x,    gates[i].y,    gates[i].tipo,
LIGHTBLUE);
       }
```

```
}

for(i=0; i<num_max; i++) gates[i].resuelta=0;









                        //PENDIENTE
                        temp=1;
                        break;
                case 4: printf("\n SALIR");
                        temp=0;
                        break;
                default:limpiar();
                        printf("Comando desconocido");
                        getch();
                        temp=1;
                        break;

                }

   }while(temp==1);

closegraph();


}
```

ANEXO 2

Librerías

comando.h

```
//Armando Nava Linares
//Datos globales para la funcion comando
int sl;
char claves[20][20]={       "nuevo"    , //0
                            "mover"    , //1
                            "valores"  , //2
                            "resultado", //3
                            "salir"    , //4
                            "nand"     , //5
                            "and"    , //6
                            "nor"      , //7
                            "or"     , //8
                            "not"      , //9
                            "exor"     , //10
                            "exnor"    ,  //11
                            "Vcc", //12
                            "GND", //13
                            "gate",//14

                                        };
//Funcion comando
//Compara una cadena con el set de comandos
//Devuelve un entero correspondiente a la posicion de un comando
//Requiere como argumento una cadena
int comando(char comando[20])
{
  int i,j,comp;

  for(i=0;i<15;i++) {
                      sl=1;
                      j=-1;
                          do{
                                  j=j++;

                                  if(comando[j]==claves[i][j])  comp=1;
                                  else if(comando[j]!=claves[i][j])  comp=0;
                                  sl=sl*comp;

                                    // if(sl==1) goto label00;

                                  }while(comando[j]!=0);
                                  if(sl==1)   goto label00;
                       }

label00:
```

```
            return(i);


}
```

limpiar.h

```
//Armando Nava Linares
//Funcion limpiar
void limpiar(void)
{

setcolor(LIGHTBLUE);
 int xmax = getmaxx();
 int ymax = getmaxy();

     line(0,0,xmax,0);
     line(0,0,0,ymax);

      line(0,ymax,xmax,ymax);
      line(xmax,ymax,xmax,0);

      line(30,0,30,ymax-100);
      line(0,ymax-70,xmax,ymax-70);
      line(0,ymax-100,xmax,ymax-100);

 //nand(3,10);
    outtextxy(30,420," nuevo   mover   valores   resultado   salir ");


 gotoxy(6,25);
   printf("                                                ");
     gotoxy(6,25);

}
```

graficas.h

```
//  graficas.h
//  Libreria que contiene las rutinas graficas de Tesis
//  Armando Nava Linares usr90@yahoo.com

int graficar(int x,int y,int tipo, int color);

/*  GRAFICAR  */

// Esta funcion requiere los parametros x,y  punto donde se
// colocara la compuerta, el tipo de compuerta a graficar,
// y el color de la compuerta
// RETORNA EL TIPO DE COMPUERTA

int nand =5;
```

```
 int and  =6;
  int nor  =7;
   int or   =8;
    int not  =9;
     int exor =10;
      int exnor=11;

int graficar(int x,int y,int tipo, int color)
{
            int i,j;

   int c_nand[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,
                       1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                          };

   int c_and[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                          };

    int c_nor[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                       0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,1,1,1,1,1,
                       0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,
                       0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                          };

     int c_or[11][25]={0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
```

```
                        0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,
                        0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                        1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                        };

   int c_not[11][25]={0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,
                        1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,1,
                        0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                        };

   int c_exnor[11][25]={0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                        1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,1,1,1,1,1,
                        0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,
                        1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,
                        0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                        };

    int c_exor[11][25]={0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                        1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,
                        0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                        1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                        0,0,0,1,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                        };
switch(tipo){

                case 5:  for(j=0;j<25;j++)
                            {
                               for(i=0;i<11;i++)
                                       {
                                         if(c_nand[i][j]==1)
```

```
                    putpixel(x+j,y+i,color);
                  }
              }
            break;
case 6:    for(j=0;j<25;j++)
           {
              for(i=0;i<11;i++)
                  {
                    if(c_and[i][j]==1)
                    putpixel(x+j,y+i,color);
                  }
              }
            break;
case 7:  for(j=0;j<25;j++)
           {
              for(i=0;i<11;i++)
                  {
                    if(c_nor[i][j]==1)
                    putpixel(x+j,y+i,color);
                  }
              }
            break;
case 8:    for(j=0;j<25;j++)
           {
              for(i=0;i<11;i++)
                  {
                    if(c_or[i][j]==1)
                    putpixel(x+j,y+i,color);
                  }
              }
            break;
case 9:  for(j=0;j<25;j++)
           {
              for(i=0;i<11;i++)
                  {
                    if(c_not[i][j]==1)
                    putpixel(x+j,y+i,color);
                  }
              }
            break;
case 10:  for(j=0;j<25;j++)
           {
              for(i=0;i<11;i++)
                  {
                    if(c_exor[i][j]==1)
                    putpixel(x+j,y+i,color);
                  }
              }
            break;

case 11:  for(j=0;j<25;j++)
           {
              for(i=0;i<11;i++)
                  {
```

```
                                        if(c_exnor[i][j]==1)
                                        putpixel(x+j,y+i,color);
                                       }
                                  }
                                break;
                        default:  break;
                }

return(tipo);
}
```

terminal.h

```
//  terminal.h
//  Libreria que contiene las rutinas graficas de Tesis
//  Armando Nava Linares usr90@yahoo.com

int term(int x,int y,int tipo, int color);
// Esta funcion requiere los parametros x,y  punto donde se
// colocara la conexion a la termina 1,2,3
// y el color de la terminal
// RETORNA EL TIPO DE terminal Vcc, GND, gate

//vcc =1;
//vcc_a  =2;
//gnd  =3;
//gnd_a  =4;

int term(int x,int y,int tipo, int color)
{
int i,j;
   int    vcc[11][10]={0,1,1,1,0,0,0,0,0,0,
                       1,0,0,0,1,0,0,0,0,0,
                       1,0,0,0,1,0,0,0,0,0,
                       0,1,1,1,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,
                       0,0,1,1,1,1,1,1,1,1,
                                                                                  };
  int  vcc_a[11][20]={0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                       0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
                    0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                                                                    };
 int   gnd[11][10]={0,0,0,0,1,1,1,1,1,1,
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,
                    1,1,1,1,1,1,1,1,1,0,
                    0,0,0,0,0,0,0,0,0,0,
                    0,1,1,1,1,1,1,1,0,0,
                    0,0,0,0,0,0,0,0,0,0,
                    0,0,1,1,1,1,1,0,0,0,
                    0,0,0,0,0,0,0,0,0,0,
                                                                    };

 int  gnd_a[11][20]={0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                    0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                                                    };
switch(tipo){
                case 1:  for(j=0;j<10;j++)
                         {
                            for(i=0;i<11;i++)
                                  {
                                    if(vcc[i][j]==1)
                                    putpixel(x+j,y+i,color);
                                  }
                          }
                         break;
                case 2:  for(j=0;j<20;j++)
                         {
                            for(i=0;i<11;i++)
                                  {
                                    if(vcc_a[i][j]==1)
                                    putpixel(x+j,y+i,color);
                                  }
                          }
                         break;
                case 3:  for(j=0;j<10;j++)
                         {
                            for(i=0;i<11;i++)
                                  {
                                    if(gnd[i][j]==1)
                                    putpixel(x+j,y+i,color);
                                  }
                          }
```

```
                    break;
            case 4:  for(j=0;j<20;j++)
                    {
                       for(i=0;i<11;i++)
                            {
                             if(gnd_a[i][j]==1)
                             putpixel(x+j,y+i,color);
                            }
                        }
                    break;

            default:  break;
        }

return(tipo);
}
```

funcion.h

```
int f_and(int, int);
int f_or(int, int );
int f_xor(int, int);

int f_not(int);
int booleano(int );

int f_and(int a, int b){  //Funcion AND
int res;

        res = (a & b);

return (res);
}



int f_or(int a, int b){   //funcion OR
int res;

        res=(a | b);

return(res);
}

int f_xor(int a, int b){ //funcion EXOR

int res;

        res=(a ^ b);

return(res);
}

int f_not(int a){   //Funcion Negacion
```

```
            int res;

    if(a==1) res=0;
    if(a==0) res=1;

return(res);
}


int booleano(int a){

int status;
            if(a==0 || a==1) status=1;
            else status=0;

return(status);
}
```

ANEXO 3

Instrucciones

Los requerimientos mínimos para utilizar el programa son los siguientes:

- PC IBM compatible con procesador 80286 o superior.
- Al menos 64K de RAM y 1 MB de memoria extendida.
- Tarjeta de vídeo VGA de 16 colores o más
- Sistema operativo Windows 3.1 o superior

Como podemos ver cualquier computadora de hoy en día sobrepasa estos requerimientos, sin embargo hay que hacer notar que el programa no corre sobre Windows XP, debido que este sistema operativo no soporta muchas aplicaciones basadas en MS-DOS. Para corregir lo anterior debe utilizarse un emulador de MS-DOS. Se hicieron pruebas utilizando el emulador DOSBox v0.63 resultando exitosas, la documentación y este emulador están en la siguiente dirección:

http://dosbox.sourceforge.net

Para ejecutar el programa basta con dar doble click sobre este, continuación aparece una pantalla esperando por alguno de los comandos disponibles mismos que se listan en la parte inferior de la pantalla y son los siguientes:

- nuevo. Una vez introducido este comando, teclear ENTER, a continuación es posible elegir una compuerta para dibujarla en la pantalla, para hacer lo anterior basta con escribir el nombre de la compuerta y dar ENTER, los nombres permitidos son: and, or, not, nand, nor, exor y exnor.
- mover. Tras escribir este comando y dar ENTER es posible elegir una compuerta para ubicarla en alguna parte de la pantalla, mediante las teclas ↑↓ (arriba y abajo) al presionar estas teclas podemos elegir la compuerta deseada, una vez elegida, presionar ENTER la compuerta cambiará de color indicando que ya es posible moverla nuevamente con las teclas ↑↓ (arriba y abajo). Cuando la compuerta esta en el sitio elegido presionar ESC y después ENTER para continuar.
- valores. Mediante este comando podemos asignar valores lógicos a las compuertas y también unirlas terminales con otras. Una ves escrito este comando dar ENTER para elegir alguna compuerta hecho lo anterior tenemos que elegir alguna de las terminales, la identificación de las terminales es la siguiente entrada superior (1), entrada inferior (2), salida (3), habiendo elegido una terminal nos aparecen las opciones que puede tomar Vcc,, GND
- resultado. Una vez colocadas las compuertas y después de hacer las conexiones deseadas, podemos usar esta opción para resolver el diagrama lógico.
- salir. Con esta opción salimos del programa.

BIBLIOGRAFÍA.

BIBLIOGRAFÍA.

- Gráficas poderosas con turbo C++.
  Keith Weiskamp, Loren Heiny.
  Limusa, 1994.

- El leguaje de programación C.
  Brian W, Kernighan, Dennis M. Ritchie.
  Prentice Hall, 2ª edición.

- Curso de programación con turbo C++.
  Luis Joyanes Aguilar.
  McGraw-Hill, 1996.

- Diseño Digital.
  M. Morris Mano.
  Prentice Hall, 1987.

www.lawebdelprogramador.com

http://www.geocities.com/joseluisdl/elmundodelcaos

http://articulos.conclase.net/libreriabgi/LibreriaBGI1.html

http://www.info-ab.uclm.es/labelec/Solar/elementos_del_pc/Ratones/principal/contenidos/interrupcion.htm

http://nayar.uan.mx/~rsilva/Ensa/Docu/El_univer/07.html

http://nayar.uan.mx/~rsilva/Ensa/Docu/El_univer/1211.html

http://ar.geocities.com/nrs_arg/pascalmania/interrup.htm

http://www.alu.ua.es/c/csv2/primeraetapa.htm

http://www.montellano.net/hist-windows.htm

www.itnuevolaredo.edu.mx/takeyas