CURSO DE PROGRAMACIÓN EN TURBO C

INDICE

INTRODUCCIÓN

Breve historia de Turbo C

El lenguaje C fué diseñado en 1972, por el científico Dennis Ritchie, en los laboratorios de Bell Telephone Inc. Con un fin específico, la creación del sistema operativo Unix, por lo que este sistema operativo tiene un entorno programable en C, el lenguaje C tuvo como predecesor al lenguaje B, desarrollado por Ken Thompson también en los laboratorios Bell, hay varias versiones de C, pero actualmente, todas ellas se apegan a la versión de C establecida por el ANSI, que se encargó de regular (como lo hace con todo lo demás) las versiones de C. De aquí nació el Estándar ANSI C (que es el que se utiliza en el entorno de Unix), un poco después, nace C++, que no es otra cosa que una mejora de C, así que todo lo que incluye C, funciona en C++, sólo que el C++ incorpora además, herramientas que permiten la P.O.O., pero para este curso, utilizaremos una versión de C hecha por Borland, es decir, Turbo C.

Fases de un programa en Turbo C

Las fases de un programa en Turbo C se pueden resumir en:

- Edición
- Compilación
- Enlazado
- Ejecución

La *Edición* de un programa consiste, simplemente, en editar el código fuente del programa, Turbo C incluye un editor para este efecto.

La *Compilación* del programa consiste en convertir el código fuente en código objeto.

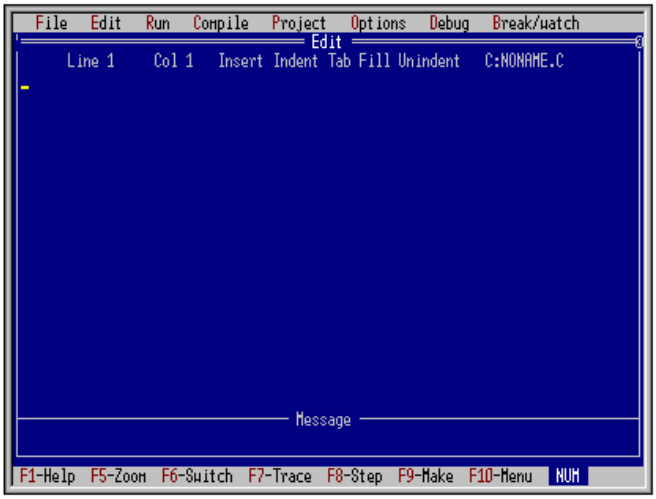
El *Enlazado* consiste en enlazar el código objeto para obtener el código ejecutable.

Finalmente, la *Ejecución* consiste precisamente en ejecutar el programa.

El entorno de Turbo C

El entorno de Turbo C facilita la edición, compilación y el enlazado de los programas, ya que incluye un editor y herramientas para compilar y enlazar los programas, para configurar el entorno de Turbo C, primero hay que conocerlo, a continuación se explican las características más importantes del entorno...

El editor de Turbo C

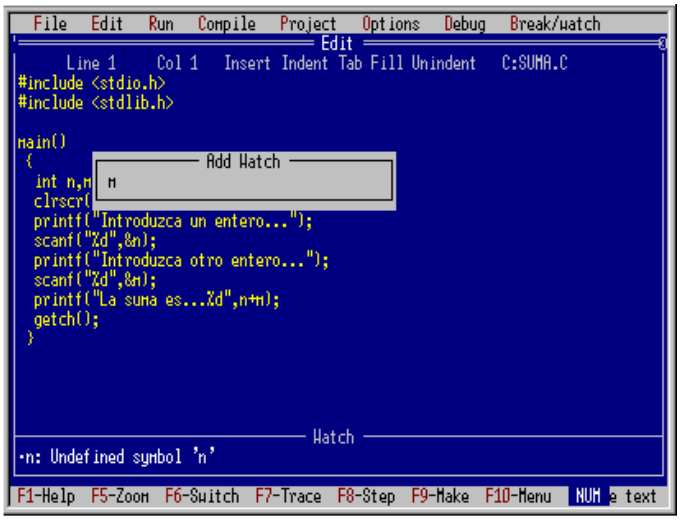


El editor de Turbo C es una poderosa herramienta que nos facilita enormemente la edición de un programa, es como un editor de texto cualquiera, pero, para los que están acostumbrados a los editores que funcionan bajo Windows, tal vez se encuentren con una forma diferente de editar texto, por ejemplo, para seleccionar un bloque de texto, en el editor de Turbo C, se presiona la combinación de teclas CTRL+K+B al inicio del bloque que se desea seleccionar, y la combinación CTRL+K+K al final del mismo. Después, con ese bloque seleccionado, se pueden hacer diversas cosas, como:

- CTRL+K+Y: Elimina el bloque.
- CTRL+K+V: Mueve el bloque.
- CTRL+K+C: Copia el bloque.

De cualquier forma, es conveniente que se familiaricen con el editor de Turbo C a medida que aprenden a programar, es decir, no es necesario aprender a usar el editor al 100%.

El debugger de Turbo C



En ocasiones, un programa que está léxica y sintácticamente bien escrito, puede no dar los resultados correctos, estos resultados pueden deberse a errores comunes de programación, tales como errores lógicos, comúnmente llamados *bugs*, aunque existen otros tipos de errores, tales como errores en tiempo de ejecución (tema del que nos ocuparemos más adelante), el debugger de Turbo C nos ayudará a detectar y corregir dichos errores lógicos. Por ejemplo, si deseamos monitorear en todo momento el valor de una variable (watch) presionaremos la combinación CTRL+F7, y posteriormente, podremos ejecutar el programa línea por línea

(F7) o función por función (F8).

Tipos de datos simples

Los tipos de datos simples en C son:

- void: Tipo de dato que no tiene valor.
- int: Para todo el rango de valores enteros*.
- float: Para todo el rango de valores reales*.
- char: Datos de tipo caracter.

Operadores

Los operadores son aquellos símbolos que nos ayudarán a relacionar y manipular los operandos, existen los operadores lógicos, relacionales y aritméticos o matemáticos.

Lógicos

AND	&&
OR	||
NOT	!

Aritméticos

Asignación	=
Suma	+
Resta	−
Multiplicación	*
División	/
División modular	%
Incremento	++
Decremento	−−

Relacionales

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

Componentes de un programa en C

La estructura básica de un programa en C es la siguiente:

#include<stdlib.h>

#include<stdio.h>

```
#define pi 3.1416

int suma(int x, int y);

char nombre[30];

/* Inicia programa principal */

main()

{

int a,b;

printf(Introduzca su nombre...);

scanf(%s,&nombre);

printf(\nTeclee un entero...);

scanf(%d,&a);

printf(\nTeclee otro entero...);

scanf(%d,&b);

printf(El resultado es%d,suma(a,b));

getch();

}

int suma(int x, int y)

{

return x+y;

}
```

La directiva #include

Por medio de esta directiva se indica al compilador cuáles son los archivos de cabecera que deberá incluir en la compilación del programa, con la finalidad de indicarle donde están los prototipos e implementación de las funciones que se utilizarán en dicho programa.

La directiva #define

A través de ella se definen las constantes y macro definiciones que se utilizarán en el programa.

Prototipos de funciones

Los prototipos de funciones no son otra cosa que una especie de declaración de funciones, donde se debe especificar el tipo de dato que devolverán dichas funciones, el nombre de la función y sus parámetros, en caso de que los tenga, aunque en Turbo C no es obligatorio el uso de prototipos de funciones, para efectos de claridad y como una práctica sana de programación, se recomienda usarlos.

Declaración de variables

Es necesario declarar las variables que se utilizarán en el programa, de modo que el compilador reserve un espacio en memoria para esas variables, la sintaxis es:

[tipo**] [**nombre_de_la_variable**] [**[dimension]**] ;** (la dimensión sólo se usa para arreglos).

Ejemplo:

int x;

int arreglo [100];

Y ya que hablamos de variables, cabe mencionar que las variables pueden ser declaradas **globales** o **locales**, solamente como referencia, las variables **globales** son aquellas variables que conservan su valor durante la ejecución de todo el programa y se declaran antes del main(), mientras que las variables **locales** solamente tienen valor durante la ejecución de la función o procedimiento en que fueron declaradas y se declaran después de la llave que indica el principio de una función o procedimiento. De cualquier manera, las diferencias entre las variables locales y globales serán objeto de estudio más adelante.

La función main()

La función main() en un programa en C significa el cuerpo del programa o el programa principal, ya que es la primer función que el enlazador busca para ejecutar; si la función main() no tiene parámetros, significa que solamente la utilizaremos para decirle al programa cuándo y cómo debe hacer las cosas, pero, si tiene parámetros, es decir, si desde la línea de comando se llama con valores de entrada, la cosa cambia, y nos ocuparemos de la función main() con parámetros más adelante.

Funciones de E/S básicas

Las funciones de E/S son las que se utilizan para capturar datos desde el teclado e imprimirlos por medio de la salida estándar (monitor). Estas instrucciones soportan ciertos formatos:

La función scanf

La función scanf captura cualquier tipo de dato introducido por el teclado, y para esto, es necesario incluir en su llamada a función el formato o tipo de dato que se pretende leer por teclado, ejemplo:

scanf(%d,&x);

En este caso, se especifica una entrada de tipo decimal o entero (%d), mientras que el operador de indirección (&) indica que se debe guardar el valor en la localidad de memoria x, en otras palabras, indica que se recibirá un valor entero y se debe almacenar en la variable x. Ahora bien, los tipos de formato más usados para la instrucción scanf son:

%d, %i	Entero decimal con signo
%f	Número real o flotante

%c	Dato tipo caracter
%s	Dato tipo cadena
%u	Sin signo

La función printf

La función printf es la contraparte de la función scanf, ya que mientras scanf lee datos desde el teclado, la función printf los escribe, es decir, provee la salida en pantalla, esta función también utiliza los formatos de scanf, con la particularidad de que printf puede modificar la salida de los datos, por ejemplo, si se declara una variable entera, y se le asigna el valor 65, y al momento de imprimir el valor de la variable se especifica una salida de tipo caracter, la salida será el caracter A (el 65 equivale a la letra A en el código ASCII).

scanf(%d,&x); /* Se lee la variable como entera */

printf(%c,x); /* Se escribe como caracter */

Delimitación de bloques de código (uso correcto de las llaves)

En la programación estructurada, a menudo se tienen que delimitar bloques de código para indicar al programa el conjunto de instrucciones que se deben ejecutar en determinado momento, para esto, se utilizan las llaves, la manera correcta de usar las llaves es la siguiente, si un bloque de código consiste en sólo una línea, las llaves no son necesarias, pero si dicho bloque de código consta de más de una línea, el uso de las llaves es imprescindible.

Estructuras de selección

Las estructuras de selección son usadas para bifurcar el control en un programa, es decir, se utilizan para evaluar expresiones que pueden tomar valores de verdadero y falso, o 1 y 0, respectivamente, ya que en C no existe el tipo de dato booleano, por lo que se usan sus equivalentes.

La sentencia if

La sentencia if es usada para evaluar una expresión lógica que puede tomar valores de 1 y 0, es decir, verdadero o falso, la sentencia if se conoce como estructura de selección simple, ya que si se cumple la condición especificada entre los paréntesis, se ejecuta un bloque de código, y si no se cumple, no se ejecuta nada, su sintaxis es la siguiente:

if (condición)

{

Bloque de instrucciones

}

Ejemplo:

if (x==0)

{

printf(El número es 0);

printf(\a);

}

En este ejemplo, si la condición (x==0) se cumple, se imprime el mensaje El número es 0 y se emite un pitido por el speaker de la computadora, y si no se cumple, pues no pasa nada.

La sentencia if − else

Esta sentencia es más o menos como la anterior, con la diferencia que en este ejemplo, si la condición se evalúa como verdadera, se ejecuta una secuencia de instrucciones, mientras que si la condición se evalúa como falsa se ejecuta otra secuencia de instrucciones; su sintaxis es la siguiente:

if (condición)

{

Bloque de instrucciones 1

}

else

{

Bloque de instrucciones 2

}

Ejemplo:

if (x==0)

{

printf(El número es 0);

printf(\a);

}

else

printf(Es un número diferente de 0);

Es posible anidar sentencias if para hacer una selección todavía más compleja, es de ir, dentri de un if puede ir otro, y dentro de éste, otro más, etc...

Ejemplo:

if (x==0)

{

printf(El número es 0);

printf(\a);

}

else

if(x<0)

printf(Es un número negativo);

else

printf(Es un número positivo);

En este ejemplo, si el número es 0, se visualizará un mensaje indicándolo, pero si no es cero, se establece una segunda condición para saber si se trata de un número negativo o positivo.

La sentencia switch − case

Esta sentencia es la utilizada para evaluar las llamadas opciones de abanico, de donde se saca una de varias opciones, es decir, switch permite múltiples ramificaciones con una sola expresión a evaluar. Es más eficiente que utilizar muchos if anidados. Un enunciado switch evalúa una expresión. Su sintaxis es la siguiente:

switch (selector **)**

{

case valor posible 1**:** acciones 1**;break**; *

**

**

**

case valor posible n: acciones n; **break**;

default: /* Opcional */

acciones a realizar si ninguna opción es la correcta

}

La sentencia **break** provoca que el control del programa dentro del switch termine, es decir, si se encuentra un valor para la expresión, se ejecutan las acciones correspondientes, y si se encuentra un break, el bloque **switch−case** termina, y las instrucciones para los otros valores correspondientes al selector se ignoran.

Ciclos

Dentro de la programación estructurada, se hace necesaria la ejecución repetitiva de tareas, lo cual no es problema, gracias a los ciclos...

El ciclo for

El ciclo for ejecuta un bloque de instrucciones n veces, es recomendable la utilización de este ciclo cuando se conoce el número de iteraciones o repeticiones de una tarea, es decir, cuando se sabe el número de veces que se va a repetir algo. Su sintaxis es la siguiente:

for(valor inicial; condición de paro; incremento o actualización**)**

{

bloque de instrucciones

}

Donde el valor inicial se asigna a una variable de tipo entero llamada variable centinela, cuya función es la de contar y controlar el número de iteraciones del ciclo.

Ejemplo:

```
for (i=0; i<10; i++)

{

printf(*);

}
```

Donde se establece como variable centinela la variable i y se inicializa con 0, la condición de paro del ciclo es que i **no** sea menor que 10, y el incremento o actualización de i es un incremento de 1 en 1, el ciclo for anterior se lee así: *se inicia i con 0, mientras que i sea menor que 10, i se incrementa en 1.*

El ciclo while

Este ciclo ejecuta un bloque de instrucciones mientras una condición determinada sea correcta. Su sintaxis es la siguiente:

while(condición**)**

{

bloque de instrucciones

}

Cuando la ejecución de un programa llega al ciclo while, sucede lo siguiente:

- Es evaluada la expresión de la *condición*.
- Si la *condición* se evalúa como falsa (es decir, a cero), el ciclo *while* termina, y la ejecución pasa al primer enunciado que se encuentre a continuación de los enunciados del while.

- Si la condición se evalúa como verdadera (esto es, 1), se ejecutan los enunciados (bloque de instrucciones) del while.
- La ejecución regresa al paso 1.

Es decir, se evalúa la expresión de la condición, y de resultar cierta, se ejecuta el bloque de instrucciones del while, mientras que, si dicha condición se evalúa como falsa, la ejecución del ciclo termina.

El ciclo do−while

El ciclo do−while es muy similar al while, es decir, también ejecuta un bloque de instrucciones mientras una condición determinada sea verdadera, pero tiene una diferencia principal con respecto al while: en el ciclo while la expresión se evalúa antes de ejecutar el ciclo, mientras que en el do−while, primero se ejecuta el bloque de instrucciones y luego se evalúa la condición. Su sintaxis es la siguiente:

do

{

bloque de instrucciones

}while(condición**);**

Tipos de datos estructurados

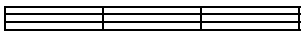
Los tipos de datos estructurados son aquellos que constan de más de un valor, como las estructuras (también llamadas tipos de datos abstractos), los arreglos y las matrices.

Arreglos

Un arreglo es una colección de posiciones de almacenamiento de datos, del mismo tipo y con el mismo nombre, pero con una dirección única, o índice. Cada posición del arreglo comúnmente es llamada *elemento del arreglo*. Los arreglos se clasifican en unidimensionales y multidimensionales, aunque hablando de multidimensionales, se usan casi exclusivamente los bidimensionales, ya que el uso de más índices o dimensiones en un arreglo hacen su uso más complejo, de cualquier forma, aquí hay una representación de los más comunes:

Arreglo unidimensional o vector simple

Arreglo bidimensional o matriz



Arreglo tridimensional

Arreglos unidimensionales

Un arreglo unidimensional es un arreglo que tiene solamente un índice, o una dimensión, comúnmente se les llama vectores, y su representación puede ser indistintamente vertical u horizontal:

Elementos

Arreglo Arreglo

5	3	7	9	1	0	2	6	9