Elementos de Programación

UNIDAD 3. EL LENGUAJE DE PROGRAMACION C

INDICE

UNIDAD 3 - El lenguaje de programación C

OBJETIVOS: *Conocer el lenguaje de programación "C". Confeccionar programas en C con sentencias de entrada, de salida, de asignación, y utilizar "funciones de biblioteca". Utilizar distintos tipos de datos simples.*

1 Un poco de Historia.

El lenguaje C fue desarrollado por Dennis Ritchie en 1972, cuando trabajaba en los laboratorios Bell, junto a Ken Thompson en el diseño del sistema operativo UNIX. Es la evolución de dos lenguajes anteriores, el primero fue el BCPL (Basic Compiled Programmaing Languaje – Lenguaje Básico de programación compilado), luego como una simplificación del lenguaje BCPL adaptado a microcomputadoras (que son las computadoras actuales) crearon el lenguaje B. Por último, el lenguaje C fue la evolución del B y de allí provienen su nombre.

A raíz de la creciente popularidad de las microcomputadoras, comenzaron a surgir numerosas implementaciones de C que diferían en parte de la definición de lenguaje original, creando pequeñas incompatibilidades y disminuyendo la portabilidad del lenguaje. La portabilidad es la capacidad de ejecutar un mismo programa en distintas computadoras (con distinto hardware) y/o distintos entornos (por ejemplo, distintos sistemas operativos). Esto hizo necesaria la búsqueda de un C estándar que fue definido por el Instituto Nacional Americano de Estándares (ANSI) en el año 1983. Por lo tanto, cuando se habla de la versión estandarizada del lenguaje se hace referencia al ANSI C.

En 1977 Kernighan y Ritchie publicaron una descripción definitiva del lenguaje "El manual de referencia del lenguaje C", que es una fuente de consulta fundamental para todo programador en este lenguaje.

Ha mediados de los 80, Bjorne Stroustrup, también en los laboratorios Bell, crea el lenguaje C++ que soporta el paradigma de programación orientada a objetos.

2 Desde el código fuente hasta el ejecutable

Como se mencionó en la unidad 1, un programa es una secuencia de instrucciones que se escriben en un lenguaje de programación que el programador entiende, pero no la computadora, por lo tanto, necesita de un proceso de traducción. El lenguaje C, es un lenguaje compilado, por lo tanto, todo el código será analizado y traducido por completo hasta obtener un programa ejecutable. Para llegar desde el código fuente hasta el programa que se puede ejecutar en una computadora hay una serie de pasos que se deben llevar a cabo. Esos pasos pueden verse en la figura 1.

Figura 1: pasos para generar un programa ejecutable partiendo del código fuente

Inicialmente se escribe el programa en el lenguaje C generando un programa fuente. El programa fuente no es más que un archivo de texto escrito respetando las sentencias del lenguaje.

Dentro del código fuente, el programador puede utilizar partes de código ya desarrolladas, de forma que las cosas repetitivas no tengan que escribirlas en cada programa. Por ejemplo, para leer datos del teclado o para mostrar mensaje por pantalla no hay instrucciones del lenguaje que puedan hacerlo en forma directa, pero, sin embargo, esa funcionalidad es parte de lo que se llaman bibliotecas de funciones. Una función es una secuencia de instrucciones que realiza una tarea específica. Las funciones se identifican por un nombre, mediante el cual desde el programa principal se "invoca" a dicha función. Invocar una función significa que el programa que se viene ejecutando en forma secuencial hace un salto, y comienza a ejecutar las instrucciones que contiene la función, cuando llega al final de la función vuelve a la línea del programa que la invoco. A las funciones se les puede pasar datos y pueden retornar un único valor. Por ejemplo, a una función que permite leer un valor de teclado se le indica en que variable va a guardar ese dato leído. A la función para mostrar un mensaje en pantalla se le envía el mensaje a mostrar, de esta forma las funciones se pueden reutilizar y adaptar a distintos programas con los datos que reciben.

Al escribir un programa si se quieren utilizar funciones ya desarrolladas se le debe avisar al compilador donde está la función que queremos utilizar, es decir, si se necesita utilizar, por ejemplo, las funciones para leer datos de teclado se le debe indicar al compilador en que biblioteca se encuentra dicha función para que la incorpore a nuestro código fuente y de ese modo podamos utilizarla. Para ello, en el código fuente se agrega una línea de código por cada una de las bibliotecas que se necesiten incluir. Las bibliotecas provistas por el entorno de programación están optimizadas de forma que cuando se incluyen el código fuente solo se incluye lo que se conoce como cabecera de las funciones. La cabecera de una función es una línea que indica el nombre de la función, que datos se le puede enviar, y si retorna o no un valor. El preprocesador es el encargado de agregar esas cabeceras de las funciones de las bibliotecas que el programador indica en el código fuente. Además, el preprocesador modifica el código fuente escrito originalmente reemplazando las constantes definidas por su valor correspondiente. Es decir, que el pre-procesador es el que genera el archivo fuente final a compilar.

El compilador es el encargado de tomar el programa fuente, chequear que cumpla con la gramática del lenguaje C y si no hay errores generar el programa objeto, que es el programa en un formato que

la computadora puede comprender. Si hay errores se informan al usuario indicando el número de línea donde se encuentra.

El programa objeto si bien ya está en un formato entendible por la computadora, aún no puede ejecutarse porque está incompleto, falta el código objeto de las funciones de biblioteca que se utilizan. Es el linker el que toma el código de dichas funciones y lo junta con el código de nuestro ***programa creando finalmente el programa ejecutable completo. Las funciones de biblioteca*** estándar están optimizadas de forma que el linker solo incorpora el código de las funciones que realmente se utilizan en el programa y NO el código de todas las funciones de la biblioteca a la cual pertenece.

3 *Estructura general de un programa en C*

Todo programa en C está estructurado en funciones. El lenguaje C tiene una función principal, que es por la cual comenzará la ejecución del programa. Dicha función se llama main, no pudiendo faltar, ya que a ésta le transfiere el control el sistema operativo al ejecutarse un programa.

Un programa en C se estructura según la imagen de la figura 2.

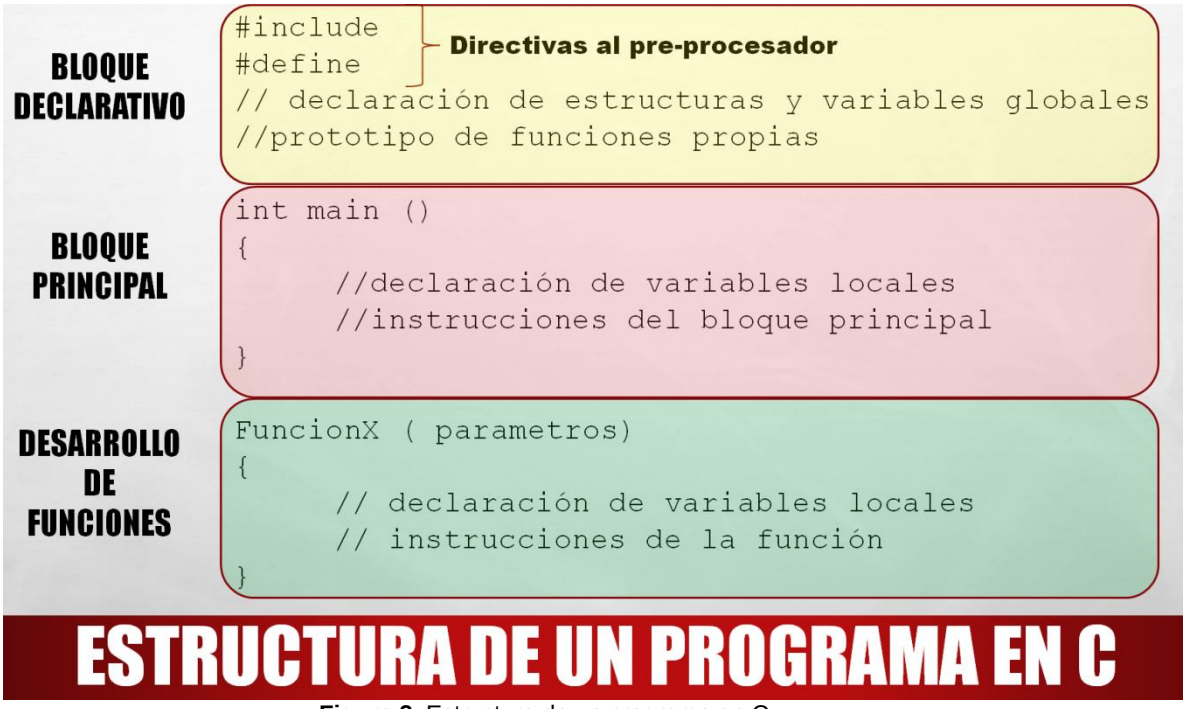


Figura 2: Estructura de un programa en C

Se analiza a continuación el contenido de los bloques enunciados anteriormente.

3.1 Bloque declarativo general

En este bloque se incorporan las directivas al preprocesador, la definición de variables globales, estructuras y prototipos de funciones propias.

3.1.1 Directivas al preprocesador

Todos los comandos al preprocesador comienzan con el símbolo #.

Si bien existen otros comandos se utilizarán solo dos: include y define.

3.1.1.1 Include

Agrega al archivo fuente el contenido de otro archivo. Se utiliza para incorporar el código de las bibliotecas al código fuente.

Para incluir bibliotecas estándar el nombre de la biblioteca se escribe entre los símbolos menor y mayor. Estas bibliotecas las busca dentro de la carpeta llamada include dentro de la estructura de directorios del compilador. Recuerde que las bibliotecas estándar solo tienen los prototipos de la función también llamados cabeceras, por lo tanto, estos archivos tienen la extensión .h que viene de header, cabecera en inglés.

La forma general es:

```
#include  <nombrebiblioteca.h>
```

Existen muchas bibliotecas que se pueden utilizar algún de ellas son:

stdio.h	para operaciones de entrada y salida.
string.h	para operaciones con cadenas de caracteres.
math.h	declara funciones matemáticas.
stdlib.h	declara funciones para conversiones numéricas, asignar memoria y otras funciones.
time.h	funciones para manipulación de fecha y hora.
conio.h	funciones de pantalla.
graph.h	incluye funciones para gráficos.
dos.h	incluye las funciones del sistema operativo DOS.
ctype.h	funciones para operar con caracteres.

También puede incluirse cualquier otro archivo que no sea una biblioteca estándar, lo que nos dará la posibilidad de crear nuestras propias bibliotecas.

Para incluir cualquier otro archivo en lugar de escribir su nombre entre menor y mayor, se utilizan comillas. Dentro de las comillas se puede escribir una ruta completa o directamente el nombre del archivo. Si solo se pone el nombre del archivo lo buscará en la misma carpeta donde se encuentra el código fuente.

Por ejemplo, #include “misFunciones.c”. Incluye todo el código de archivo misFunciones.c en el programa fuente. Dicho archivo DEBE estar en la misma carpeta que archivo fuente sobre el que estamos trabajando sino no lo encontrará.

En cambio, #include “c:\\funciones\\misFunciones.c” buscará el archivo a incluir en el disco c dentro de la carpeta funciones. Observe que se utilizan dos barras en lugar de una, esto es porque la barra es un carácter llamado de escape utilizado para realizar funciones especiales dentro del lenguaje, esto se verá a continuación.

Siempre es recomendable para mayor portabilidad utilizar la primera opción y poner los archivos a incluir en la misma carpeta que el código original.

3.1.1.2 Define

Permite definir un texto a reemplazar dentro del programa. Se utiliza para definir constantes y macros.

Forma general es:

```
#define  nombre-simbólico   texto de reemplazo
```

A partir de esta instrucción cualquier aparición del nombre-simbólico, dentro del programa, será reemplazada por el texto de reemplazo. Permite definir constantes globales ya que la etiqueta será buscada y reemplazada en todo el programa, siempre teniendo en cuenta que se trata de un reemplazo de texto, es decir, actúa como el comando buscar y reemplazar de cualquier editor de texto. El programa a compilar será el resultante luego de realizar los reemplazos.

Ejemplos:

```
#define MAXIMO 100

#define PI  3.14159

#define TITULO "RESUMEN"
```

También puede definirse lo que se conoce como macros haciendo que la etiqueta sea reemplazada por varias instrucciones incluso pudiendo enviarle datos para hacerlo más genérico. Las macros no se utilizarán en esta materia.

3.1.2 Definición de variables globales y estructuras

Al escribir un programa, para guardar los datos con los que trabajará, necesita variables. Dentro de los lenguajes de programación, las variables tienen lo que se conoce como "ámbito" de vigencia de la variable. El ámbito indica desde que parte de nuestro programa podemos acceder a esa variable. En general, cada bloque o función del programa define sus propias variables, lo que se conoce como variables locales. Si se necesita una variable que pueda ser accedida desde cualquier parte del programa se deberá declarar en forma global, antes del bloque principal. No es recomendable la utilización de variables globales ya que fácilmente puede perderse el control de donde fue modificada, y además, hace mucho más difícil que el código realizado pueda reutilizarse en otros programas.

Más adelante se verá el tema de estructuras de datos, en general una estructura deberá ser accedida desde todo nuestro programa por lo que su declaración será en el ámbito global.

3.1.3 Prototipos de funciones propias

Además de poder utilizar funciones de bibliotecas ya desarrolladas, más adelante se construirán funciones propias. En el bloque declarativo se incluyen lo que se conoce como prototipo de funciones, que es la especificación del nombre de la función, que datos recibe y que retorna. El código con las instrucciones de dichas funciones se pondrá debajo del bloque principal.

El prototipo de funciones se pone para que el programa quede más ordenado y poder encontrar rápidamente el bloque principal de nuestro programa. También es posible no poner los prototipos y escribir la función completa antes del programa principal, pero esto haría más difícil ubicarse dentro del código fuente. Si dentro de una función se llama a otra función, debe ponerse primero el prototipo de la que es llamada, y luego, el prototipo de la función que la llamó.

3.2 Bloque Principal

Es el inicio del programa. Como el lenguaje C está estructurado en funciones, esta es la función principal que siempre debe estar presente y es lo primero que se ejecuta. El formato general es el siguiente:

```
int main ()
{

    return 0;
}
```

Donde main es el nombre de la función (main significa principal en ingles). La palabra int significa que esta función retorna un número entero. Todo programa debe retornar al sistema operativo un número entero que indica si la ejecución fue exitosa o no, retornar un 0 significará que no hubo errores y cualquier otro valor servirá para indicar error. Los paréntesis vacios indican que la función no recibe datos, en materias siguientes se trabajara con datos enviados al programa al iniciar el mismo. También para indicar que no recibe datos en lugar de los paréntesis vacios puede ponerse dentro la palabra void, que es el tipo de dato vacio, indicando que no recibe nada.

Las llaves forman el bloque de la función principal, dentro de las llaves se escribirán todas las instrucciones del programa, comenzando siempre por la declaración de variables locales que se verán en la sección siguiente.

Antes de cerrar la llave del bloque del programa principal se pone la palabra return y en este caso se retorna un 0 indicando al sistema operativo que el programa finalizó correctamente.

3.3 Desarrollo de Funciones Propias

Si se utilizan funciones propias y en el bloque declarativo se incluyó el prototipo de las mismas, debajo de la función principal se escribe el código completo de dichas funciones. Cada una de ellas tendrá su nombre, datos que recibe, valor de retorno y su bloque definido entre llaves. Nuevamente dentro de cada función al comienzo del bloque se definen las variables que se utilizan localmente en la misma, y luego, las instrucciones para realizar la funcionalidad deseada.

4 Elementos del lenguaje

Todo lenguaje de programación está compuesto por una serie de elementos que se detallan a continuación y son los elementos básicos que componen el C:

1. Alfabeto
2. Identificadores
3. Constantes
4. Variables
5. Tipos de datos
   - números enteros o de punto fijo
   - números reales ó de punto flotante
   - caracteres – cadenas de caracteres – string
   - modificadores de tipo
6. Sentencias
7. Reglas de puntuación

4.1 Alfabeto

Como en cualquier lenguaje este consta de:

- Letras mayúsculas y minúsculas (son distintas MAYUSCULAS y minúsculas)
- Dígitos del 0 al 9
- Caracteres especiales: () { } ; : " ', . & * % #
- Palabras reservadas: main, for, while, include, If, switch, auto, etc. (estas palabras se llaman reservadas ya que no pueden ser utilizadas como identificadores)
- Operadores (aritméticos, de relación, lógicos, de incremento, de asignación, etc.). Ej.: + *, > <, &&, ++, =

4.2 Identificadores

Se llama así, a los nombres que se utilizan para nombrar a las constantes, variables de cualquier tipo, y a las funciones.

Puede ser una sucesión de letras, dígitos y algunos caracteres especiales, comenzando siempre con letra. Se admite como letra al carácter _ (guion bajo). No hay límites en cuanto a cantidad máxima, como mínimo una letra.

Ejemplos:

a A suma x33 G lote superficie AutoLuz k_k23

4.3 Constantes

Son aquellos valores que no se alteran durante la ejecución del programa. Hay dos tipos de constantes, según su contenido, numéricas (pueden escribirse en sistema decimal, octal y hexadecimal) y NO numéricas. Por ejemplo:

- Numéricas: 56, 5, 8.90E12, -1.25, 3.1415
- No numéricas: '$', "mts", "Grados", ´5´, "34-35"

Las constantes pueden utilizarse como parte de operaciones, por ejemplo, si a la variable suma se le quiere sumar diez entonces se puede escribir:

suma = suma +10;

En este caso 10 es una constante numérica entera.

También pueden definirse contantes con un identificador de dos formas:

- En el bloque declarativo usando el define, visto anteriormente.
- Anteponiendo al tipo, en la declaración de variables, la palabra const.

4.4 Variables

Se define una variable, como la representación simbólica de un lugar de la memoria, donde se guarda un valor, que es el valor de la variable. Su nombre es un identificador, cuyo valor se puede alterar durante la ejecución de un programa.

A cada variable se le debe asignar un tipo de datos e inclusive, se le puede asignar un valor inicial, como se verá luego.

Debemos tener clara la diferencia entre letras minúsculas y mayúsculas, ya que se asumen como caracteres distintos en el lenguaje C.

4.5 Tipos de dato

Una de las características más importantes del lenguaje C es su capacidad para operar con una gran diversidad de tipos de datos, admitiendo la creación por parte del usuario de nuevos tipos.

Los algoritmos operan sobre datos, los cuales pueden ser de distinto tipo y se caracterizan por:

- Un rango de valores posibles
- Un conjunto de operaciones realizables sobre dicho tipo de datos
- Su representación interna.

Significa que al asumir un tipo de dato se indica la clase de valores que pueden tomar las variables, y las operaciones que se pueden hacer sobre ellos.

Por ahora se analizarán los tipos de datos "simples", que son aquellos donde una variable ocupa "un lugar de memoria" y contiene un único valor.

Los tipos de datos simples más comunes se muestran en la tabla 1.

Tabla 1: Tipos de dato simples

Tipo	Nombre	Espacio que ocupa	Rango
Entero corto	short int	2 bytes	-32768 a 32767
Entero	int	4 bytes	-2.147.483.648 a 2.147.483.647
Caracter	char	1 byte	-128 a 127
Real	float	4 bytes	$1.7\ 10^{-38}$ a $1.7\ 10^{38}$ (7 díg.)
Real doble	double	8 bytes	$1.7\ 10^{-308}$ a $1.7\ 10^{308}$ (16 díg.)

Si no se necesitan utilizar números negativos, al tipo de dato sin decimales, puede anteponerse el modificador unsigned de forma que el rango ya no se divide entre positivos y negativos sino que toma solo números positivos quedando los rangos según la tabla 2.

Tabla 2: Modificador unsigned

Tipo	Nombre	Espacio que ocupa	Rango
Entero corto sin signo	unsigned short int	2 bytes	0 a 65535
Entero sin signo	unsigned int	4 bytes	0 a 4.294.967.295
Carácter sin signo	unsigned char	1 byte	0 a 255

El lenguaje también dispone de tipos de datos ESTRUCTURADOS, que como su nombre lo indica, cada variable puede tener una adecuada estructura conteniendo una serie de valores de igual ó distinto tipo.

Ejemplos: cadenas, arrays, estructuras, etc.

4.5.1. Números enteros

Son aquellos que solo contienen dígitos, precedidos o no por el signo. No tienen exponente (E) ni punto decimal.

Ejemplos:

- En decimal: 0 32767 -12345 231 89
- En octal: Se antepone al nro un 0 (cero). Ej. 071 en octal es igual al 57 en decimal
- En hexadecimal: Se antepone al nro. 0X. Ej.: 0x100 en hexadecimal es igual al 256 en decimal

4.5.2. Números reales

Pueden escribirse de forma diferente, incluyendo dígitos, “un punto decimal”, y opcionalmente, pueden incluir un signo y/o un exponente.

Ejemplos:

12.3 -9.0009 3.0E12 4E-15 -2.5E-12 0.235

4.5.3. Caracteres

Las variables del tipo char pueden guardar un solo símbolo, un carácter. Pero en realidad internamente guarda un número que corresponde al código ASCII (sigla en inglés de American Standard Code for Information Interchange - Código Americano Estándar para el Intercambio de Información). Por ejemplo, el código ASCII de la letra A es el 65 el de la B el 66, y así sucesivamente. El código de las letras minúsculas comienza en el 97 para la letra a.

Por lo tanto, una variable del tipo char se puede utilizar para guardar un carácter o números pequeños, dependerá luego de cómo mostremos dicha información.

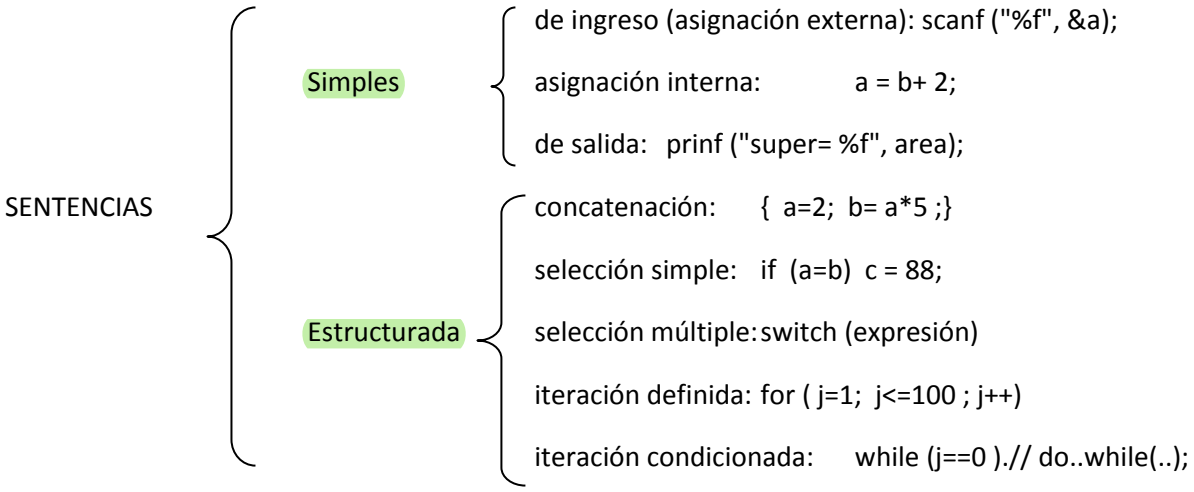
Las constantes del tipo char se escriben entre apostrofes (o comillas simples)

Ejemplos: 's' 'v' '*' '@' '9'

Es importante destacar que las letras tienen números correlativos lo que servirá más adelante para realizar validación de los datos ingresados.

4.6 Sentencias

Una sentencia en C puede ser una sola instrucción ó un grupo de instrucciones encerradas entre llaves. Hay dos tipos de sentencias, las simples y las estructuradas. Más adelante se verá en detalle cada una de ellas.



4.7 Reglas de puntuación

A medida que se avance en del desarrollo de los temas se verá puntualmente la forma correcta de escribir cada una de las sentencias y bloques del lenguaje. Como regla general debe tener en cuenta que:

- Las instrucciones finalizan con; (punto y coma)
- Los bloques se delimitan entre { } (llaves)
- Debe respetarse las minúsculas y mayúsculas ya que el lenguaje es sensible, teniendo en cuenta que la mayoría de palabras reservadas se escriben todas en minúscula.
- Si dentro del código fuente quieren poner comentarios se utiliza:
  - // para los comentarios de una sola línea.
  - /* comentario */, para comentarios de varias líneas.
  - Los comentarios se utilizan para clarificar el código fuente agregando notas que expliquen lo que se está haciendo. Los comentarios no agregan peso al programa final ya que son eliminados por el pre-procesador. Solo quedan en el código fuente.

5 Declaración de variables

Todas las variables que se utilicen en el programa (main) deben ser declaradas al inicio del mismo o de cada función que las use, asignando valores iniciales, si se requieren. Esta declaración tiene por objeto asociar a cada variable un tipo de dato y efectuar una reserva de memoria.

Forma general es:

tipo de dato lista de variables;

En realidad, antes del tipo de dato en el lenguaje C se especifica donde se va a guardar dicha variable. Pero en esta materia se usa solo el almacenamiento por defecto y por lo tanto no se escribe. Es equivalente de escribir la palabra `auto` delante del tipo de dato para indicar el tipo de almacenamiento automático. Es por eso que `auto` es una palabra reservada del lenguaje.

Ejemplos:

- Para definir una única variable del tipo entera llamada n1:

```
int n1;
```

- Para definir tres variables enteras:

```
int num1, num2, suma;
```

- Para definir dos variables reales, el segundo inicializado en un valor:

```
float a, b=7.2;
```

- Para definir dos variables char, el primero inicializado en un valor:

```
char letra='s', símbolo;
```

A la declaración de tipo se le puede anteponer la palabra `const`, en cuyo caso no serán variables sino constantes y luego se asigna el correspondiente valor.

Ejemplo:

```
const float pi = 3.14159;
```

Esto significa que pi ocupa un lugar en memoria, pero su valor al ser constante se asigna por única vez y ya no puede ser modificado en todo el programa.

AMBITO DE INCUMBENCIA: Las variables según el lugar en el cual se las declare pueden ser:

- **VARIABLES GLOBALES.** Son las definidas en el bloque declarativo general y su campo de validez es todo el programa, incluyendo todas las funciones.
- **VARIABLES LOCALES.** Son las definidas dentro de cada función, incluso en el main (), y su campo de validez se limita solo a la función donde fueron definidas. Cabe destacar que es posible crear variables locales con el mismo identificador en distintas funciones, teniendo siempre en cuenta que a pesar de tener el mismo nombre son variables distintas ya que cada función define sus variables locales en áreas separadas de memoria. También dentro de una función es posible definir un bloque interno (encerrado entre llaves) pudiendo definir variables locales a ese bloque, pero pudiendo acceder a las variables del bloque de la función que lo contiene.

6 Funciones de entrada / salida

El lenguaje C no posee sentencias con capacidad de generar la entrada y/o salida de datos, por lo cual se recurre a funciones de biblioteca. Estas funciones permiten la transferencia de información entre los dispositivos de entrada y/o salida con la computadora. Se puede invocar a estas funciones desde cualquier punto del programa. Las que más se utilizarán son scanf () y printf (), que son sentencias con formato, ambas están incluidas en la biblioteca stdio.h.

6.1 SALIDA - función printf ():

Permite la transferencia de información entre la computadora y un dispositivo de salida como ser la pantalla o la impresora.

Forma general es:

```
printf ( "cadena de control", lista de argumentos );
```

6.1.1 Cadena de Control

La cadena de control: se compone de una lista de:

1. Especificaciones de formatos
2. Secuencias de escape
3. Leyendas, títulos ó mensajes aclaratorios

Con estos elementos entre comillas puedo diseñar la salida que deseo exhibir.

1) Especificaciones de formato.

Se debe indicar una especificación de formato para cada variable a informar, el cual se compone del símbolo % seguido del carácter de conversión. Indica al compilador que tipo de variable va a ser exhibida. Los códigos de formatos más utilizados son:

% d para un número entero con signo

% f para un número real

% c para un carácter

% s para una cadena de caracteres (un texto)

% e para números reales con exponente

% hd para entero corto (short int)

% x para un entero en hexadecimal – imprime en hexadecimal.

Se puede formatear la exhibición, o sea, especificar "la longitud de campo deseada" para cada valor a exhibir (un valor entero luego del %). Esta longitud puede precederse con un signo -, lo cual significa alinear por izquierda, con + o sin poner nada alinea por derecha. Si tengo números reales, especifico el total de posiciones, un punto y la cantidad de cifras decimales.

Ejemplos

%4d: muestra un número entero en cuatro lugares, si el número tiene menos de cuatro deja espacios en blanco a la izquierda, si tiene más lo muestra igual completo.

% 8.3 f: muestra un número real que en total ocupa 8 espacios con 3 decimales

%.0f (sin decimales): muestra un número real sin decimales.

Los datos tipo char – como enteros

Una variable definida "tipo char" también puede ser impresa con un formato entero. Dependiendo del formato que se utilice para imprimirla, ya sea %c ó %d, se obtienen distintos resultados.

La siguiente línea de código muestra la constante de la letra a con su valor en decimal (código ASCII) y en formato caracter

```
printf ("El caracter %c tiene el valor %d ",'a', 'a');
```

Muestra en pantalla: El carácter a tiene el valor 97

2) Secuencias de escape.

Luego de mostrarse un valor, el cursor queda después del último carácter exhibido, en la misma línea. Si quiero que vuelva al principio de la próxima línea debo colocar antes de la comilla de cierre el "carácter de escape" \n. Puede colocarse en cualquier lugar de la cadena de control. Existen otras secuencias de escape, están formadas por una \ y un carácter que puede ser una letra o una combinación de dígitos. Se usan para tabular, cambio de línea y representar caracteres no imprimibles.

SECUENCIAS	ACCION
\n	nueva línea
\t	tabulación horizontal
\"	imprime comillas
\'	imprime un apóstrofe
\b	retrocede un espacio el cursor
\r	retorno de carro

3) Leyendas, títulos ó mensajes aclaratorios.

Todo texto que se ponga entre las comillas de la cadena de control que no sea un formato o una secuencia de escape se verá tal cual, en pantalla, por lo que pueden escribirse mensaje para el usuario. Hay que aclarar que los caracteres especiales del español como la letra eñe y acentos no se visualizan correctamente.

6.1.1 Lista de argumentos:

Es una lista de expresiones, constantes y variables, cuyos valores se van a imprimir, separadas por comas y en concordancia con los formatos especificados en la cadena de control.

Ejemplo 1 :

```
int codi=7 ; float temp= -31.42;

printf ("\n El codigo es %4d y la temperatura es = %7.2f \n", codi, temp);
```

Comienza cambiando el cursor de línea y luego imprime:

El codigo es 7 y la temperatura es = -31.42

Luego el cursor baja una línea.

Ejemplo 2

```
printf ("31+21 es igual a: %d cms", 21+31);
```

Imprime: 31+21 es igual a: 52 cms

Ejemplo 3:

```
printf("chau %c  seis = %d", '@',6 );
```

Imprime: chau @ seis = 6

6.2 ENTRADA - función scanf ()

Se utiliza para el ingreso de información a la computadora, a través, de las variables indicadas. Es la encargada de leer los datos que se ingresan desde el teclado. Al encontrar esta función, la computadora queda a la espera del ingreso de uno ó más valores desde el teclado, luego de ingresar el último dato, debo pulsar "enter ". Si hay más de un dato a digitar, se pueden separar con espacio, una tabulación o con enter, pero siempre al final se debe presionar enter.

Forma general:

```
scanf ("formatos", lista de argumentos);
```

Donde el primer parámetro se compone de una lista de especificaciones de formatos entre comillas. Se debe indicar una especificación de formato para cada variable que se quiera leer. Son los mismos que se utilizan para el printf(...).

Es importante no dejar espacios ni poner otros caracteres entre los formatos para las variables para evitar problemas al leerlos.

La lista de argumentos está formada por la dirección de memoria de las variables que se desean leer. Para obtener la dirección de memoria de una variable se utiliza el operador &. Este operador indica el lugar de memoria donde será enviado el valor ingresado por teclado y registrado por la función, mediante el nombre del identificador. Se debe respetar el orden de los formatos según las direcciones de memoria que se especifiquen en el argumento, el primer formato corresponderá con la primera dirección de memoria, el segundo con la segunda y así sucesivamente.

Ejemplo1: Leer un entero:

```
int a;
scanf("%d", &a);
```

Ejemplo 2: Leer dos enteros:

```
int n1, n2;
scanf("%d%d",&n1,&n2);
```

Ejemplo 3: Leer un real y un entero:

```
int num;
float f;
scanf ("%f%d", &f, &num);
```

Todos los datos al ser ingresados por teclado se almacenan temporalmente en un buffer de teclado, que es una posición de memoria intermedia que guarda todas las teclas presionadas. Luego, la función scanf toma los datos de ese buffer y los pasa a la dirección de memoria indicada según el formato establecido. Este procedimiento funciona perfectamente para variables numéricas pero el scanf tiene un problema al leer caracteres. El formato %c lee un solo carácter, por lo tanto, lo va a buscar al buffer de teclado, pero el buffer muchas veces no está vacío ya que si antes se leyó otro valor el código de la tecla enter que se presiona para confirmar los datos queda en el buffer ya que no es parte del dato ingresado. Este código de la tecla enter es tomado automáticamente por el scanf, y, por lo tanto, no permite ingresar el dato. Entonces para poder leer un carácter de teclado si antes se leyó otro dato hay que limpiar el buffer de teclado. Para ello hay dos formas:

- Usando la función fflush(stdin): esta función limpia el buffer de la entrada estándar, es decir, del teclado. El problema es que esta no es una función estándar por lo que no funciona en todos los compiladores.
- Quitar el caracter que sobra del buffer utilizando getchar (): getchar es una función que recupera un carácter del buffer, por lo tanto, se puede poner luego de leer un dato para "limpiar" ese carácter del enter que quedó en el buffer.

Ejemplo de un programa completo que lee primero un dato entero y luego una letra, limpiando el buffer en el medio:

```
#include <stdio.h>
int main ()
{
    int num;
    char letra;
    printf ("Ingrese un numero: ");
    scanf("%d", &num);
    getchar();
    printf("Ingrese una letra: ");
    scanf("%c",&letra);
    printf("Se ha ingresado el numero: %d y la letra: %c", num, letra);
    return 0;
}
```

7 Sentencias de asignación y aritméticas

Las sentencias secuenciales se escriben directamente tal como se explicó en la unidad 2, tanto para operaciones matemáticas como para asignaciones. La única diferencia con el diagrama es que en el código C cada instrucción al final se termina con el punto y coma.

7.1 Combinación de operadores / operadores especiales

El C lenguaje admite la simplificación de una variable en las sentencias aritméticas cuando se combinan los operadores aritméticos y el igual.

Lo común es:

variable= variable operador expresión;

Se puede reemplazar por:

variable operador= expresión;

Algunas combinaciones posibles son:

* =	mult. + asignación
/ =	división + asignación
% =	resto + asignación
+ =	suma + asignación
- =	resta + asignación

Ejemplos:

x = x + 3;	se puede escribir:	x += 3 (suma y asignación)
x = x * 3;	se puede escribir :	x * = 3 (multiplica y asigna)

El lenguaje C dispone además de dos operadores especiales el ++ y el --.

El operador ++ permite incrementar en 1 el valor de una variable, y el operador—decrementar en 1 el valor de una variable.

Por ejemplo, a= a +1; puede escribirse como a++; pero también, puede escribirse como ++a; o incluso usando la combinación de operadores como a +=1;

El operador ++ o -- puede usarse tanto antes como después de la variable. Si se la aplica solo a la variable tiene el mismo efecto pero en una asignación el efecto es diferente. Por ejemplo, el siguiente programa:

```
#include <stdio.h>
int main()
{
    int n1=5,n2;
    n2=n1++;
    printf("n1:%d  n2:%d", n1,n2);
}
```

Mostrará: n1: 6 n2: 5

Ya que el operador ++ se toma como un post-incremento y se aplica luego de asignar el valor de `n1` a `n2`. En cambio, si esa línea se reemplaza por `n2=++n1;`

Mostrará: `n1: 6 n2: 6`

Ya que el operador ++ se toma como un pre-incremento y se aplica antes de asignar el valor a n2.

Esto mismo aplica para el operador --

7.2 Evaluación de las sentencias de asignación

Se realiza en tres pasos:

- Se evalúa la expresión, o sea, se la resuelve obteniendo un resultado.
- Si es posible, adecua el tipo de dato del resultado de la expresión al tipo de dato de la variable de la izquierda. Si no es posible puede o no indicar error y asignar cualquier valor (entero=real: asigna parte entera).
- Asigna el resultado de la expresión a la variable de la izquierda.

PRECAUCIONES:

No colocar dos operadores consecutivos, salvo los permitidos, usar paréntesis si es necesario o existen dudas sobre la precedencia de los operadores.

La división por cero es una operación prohibida, cancela el programa.

7.3 Pausa para mostrar los resultados

Los programas que se realizarán son programas de consola. Si se trabaja con el sistema operativo Windows al ejecutar desde la interfaz gráfica un programa de consola (haciendo doble click al ejecutable) el mismo se ejecuta, pero al finalizar automáticamente se cierra y no permite ver los resultados. Por lo tanto, es necesario antes de finalizar el programa poner una pausa para permitir al usuario visualizar los resultados. Para ello podemos utilizar:

- La función getch () disponible en la biblioteca conio.h, esta función es similar al getchar, pero no muestra el caracter presionado en pantalla. El problema es que getch () no es una función estándar, por lo tanto, no funciona con todos lo compiladores.
- Hacer una llamada al sistema operativo e invocar la pausa. En Windows la instrucción sería: system("pause"); y para poder usar la función system debe incluirse la biblioteca stdlib.h.
- Poner un mensaje propio personalizado y esperar a que se presione una tecla cualquiera (usando getchar o scanf) o una tecla particular validando el ingreso de datos utilizando sentencias repetitivas que se verán en unidades siguientes.

8 Programa de ejemplo

Siempre existen diversas formas para realizar un programa. Puede cambiar desde el nombre que se les den a las variables hasta las instrucciones que se utilicen, por lo tanto, es posible que nuestros programas nunca queden exactamente iguales al de un compañero o al del profesor, pero aún así siguen estando correctos. Vea un sencillo ejemplo y distintas formas de resolverlo:

Programa a realizar: Ingresar dos números enteros, calcular y mostrar la suma:

Primera forma: con dos scanf y una variable para el resultado

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
     int n1,n2, suma;
     printf("Ingrese el primer numero a sumar:");
     scanf("%d", &n1);
     printf("Ingrese el segundo numero a sumar:");
     scanf("%d", &n2);
     suma = n1+n2;
     printf ("El resultado de la suma es:  %d\n", suma);
     system("pause");
     return 0;
}
```

Segunda forma: con un solo scanf y una variable para el resultado

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
     int n1,n2, suma;
     printf("Ingrese dos numeros a sumar:");
     scanf("%d%d", &n1, &n2);
     suma = n1+n2;
     printf ("El resultado de la suma es:  %d\n", suma);
     system("pause");
     return 0;
}
```

Tercera forma: con un solo scanf y sin variable de resultado:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
     int n1,n2;
     printf("Ingrese dos numeros a sumar:");
     scanf("%d%d", &n1, &n2);
     printf ("El resultado de la suma es:  %d\n", n1+n2);
     system("pause");
     return 0;
}
```