LENGUAJE

DE

PROGRAMACIÓN

UNA BREVE INTRODUCCIÓN.

Universidad Tecnológica de la Mixteca
Instituto de Electrónica y computación.
Elaboró: Carlos Alberto Fernández y Fernández.

Huajuapan de León, Oaxaca; a 19 de Octubre de 1998.

Tabla de contenido

0. Introducción. ____ *4*

¿Porqué empezar de cero? ____ **4**

Breve historia de C. ____ **4**

Características de C. ____ **5**

1. Datos. ____ *6*

Hola mundo. ____ **6**

Tipos de datos en C. ____ **6**

Declaraciones. ____ **7**

Operadores aritméticos. ____ **7**

Operador de asignación. ____ **7**

Operadores relacionales. ____ **7**

Operadores lógicos. ____ **8**

Operador condicional. ____ **8**

Operador coma. ____ **8**

2. Control. ____ *9*

Estructuras condicionales. ____ **9**

Decisión simple (if). ____ 9

Decisión doble (if-else) ____ 9

Decisión múltiple (switch) ____ 10

Estructuras iterativas. ____ **10**

Estructura iterativa while. ____ 10

Estructura iterativa for. ____ 11

Estructura iterativa do-while. ____ 12

3. Funciones. ____ *13*

La instrucción return. ____ **14**

Prototipo de funciones. 14

Llamada de funciones. 14

Ejemplo de uso de funciones en C. 15

4. Arreglos. *16*

Inicialización de un arreglo. 16

Uso de arreglos. 16

Ejemplo de un programa con uso de arreglos en C: 17

5. Apuntadores. *18*

Operadores de apuntador. 18

Operadores aritméticos para apuntadores. 19

Apéndice A. Funciones del Lenguaje C más comunes. *20*

6. Bibliografía. *21*

Lenguaje de programación C

0. Introducción.

¿Porqué empezar de cero?[1]

No pude evitar la tentación de empezar de cero debido al apego que tiene el lenguaje C por el cero:

- C usa el cero para indicar un valor de falso y diferente de cero para verdadero.
- El valor más bajo en el índice de un arreglo es cero.
- Las cadenas en C finalizan con un valor de cero.
- Los apuntadores usan el valor de cero para indicar un valor nulo (NULL).
- Las variables externas y estáticas son inicializadas en cero por omisión.

Por esas razones y por pretender que este sea un documento básico de C - que empiece de cero en el lenguaje, si ahondar en conceptos básicos de programación - y no un manual de referencia completo.

Breve historia de C.

No podemos empezar de cero si no contamos un poco de la historia de C. El lenguaje C es un lenguaje de propósito general desarrollado en los Laboratorios Bell en 1972. Sus creadores son Dennis Ritchie[2] y Ken Thompson. Es un lenguaje del estilo de Algol y Pascal. Aunque en realidad surge para cubrir las carencias del Lenguaje B - basado a su vez en BCPL- , desarrollado en 1967 por Martin Richards. B era un lenguaje con un manejo de tipos muy débil. C entonces incorpora algunas ideas de B pero hace más fuerte el control de tipos, le agrega definiciones de estructuras y algunos operadores extra, entre otras cosas.

Estos cambios era necesarios principalmente por los primeros usuarios de UNIX. Posteriormente se uso C para desarrollar el kernel de UNIX y da ahí se empezaron a desarrollar el resto de las aplicaciones de UNIX en C.[3]

En 1980 C cobra mayor popularidad con el surgimiento de versiones comerciales del lenguaje. El grupo ANSI en 1982 desarrolla "ANSI C" la versión estándar de C propuesta por el grupo. C se convierte en el lenguaje de más aceptación por parte de los programadores. Posteriormente, ANSI desarrolla C++ que incorpora los conceptos de la programación orientada a objetos siendo compatible con la versión de C estándar.

Se dice que C y C++ reúnen al mayor número de programadores en el mundo. Aunque Java parece estar acercándose rápidamente, pero esa es otra historia...

[1] Debo decir que no es una idea original, pero que me resultó un detalle curioso cuando tuve entre mis manos por primera vez el libro "A Book on C" [2]
[2] Coautor del libro "El lenguaje de Programación C" [1]

Características de C.

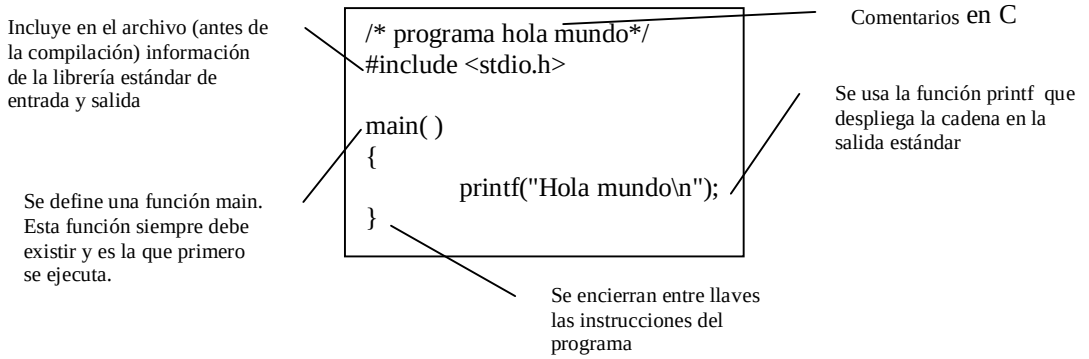
A continuación presento algunas características que hicieron a C el lenguaje favorito de muchos desarrolladores [2]:

- Es un lenguaje de tamaño reducido.
- Es el lenguaje nativo de UNIX.
- Es portable, o al menos más que algunos lenguajes de su época.
- Permite la programación modular.
- Es la base de C++.
- Corre eficientemente en la mayoría de las plataformas.

1. Datos.

Hola mundo.

Para empezar se muestra un primer programa en C, del que no pude evitar escribir la clásica frase "hola mundo":



A partir del ejemplo anterior podemos mencionar algunos puntos:

- Los comentarios sirven para documentar un programa, explicando el funcionamiento del mismo, incluyendo información específica sobre alguna operación u otras cuestiones. De esta forma es más fácil para el programador revisar un programa si éste se encuentra debidamente "comentado".
- La instrucción #include, le dice al preprocesador de C que agregue el archivo stdio.h antes de iniciar el proceso de compilación, de manera que se tengan disponibles para este programa las funciones de entrada/salida estándar.
- La función main es la punta de lanza de los programas en C. Es por esto que esta función siempre debe de existir en un programa. En este ejemplo, la función no incluye argumentos ni tiene asignado un valor de retorno, pero podría darse el caso dependiendo del contexto del problema que fuera necesario manejar argumentos o algún valor asignado al nombre de la función.
- La instrucción printf es una función que envía a la salida estándar la cadena "Hola mundo", que forma parte de la librería stdio.h. Es por el uso de está función que es necesario agregar el #include <stdio.h> para que la función este disponible. La función no forma parte del lenguaje.
- Las llaves {} se usan para encerrar bloques de instrucciones. Es necesario encerrar las instrucciones que forman parte de una función ó de más de una línea de código en una estructura de control. Algo similar al *begin* y *end* de Pascal.

Tipos de datos en C.

Los tipos de datos básicos o simples de c son:

- Char. Almacena un único carácter (a-zA-Z0-9!"#$...) y ocupa un byte de espacio en memoria.
- Int. Para el manejo de enteros, generalmente ocupa 2 bytes pero en última instancia depende del compilador.
- Float. Para valores de punto flotante o números reales, ocupa 4 bytes de almacenamiento.
- Double. Almacena números de punto flotante de doble precisión y requiere 8 bytes de memoria.

- Void. Ocupa cero bytes y no tiene valor.
- Como un tipo especial se puede considerar el apuntador que almacena la dirección de una celda de memoria.

Declaraciones.

En C, las variables deben de ser declaradas para poder ser usadas. La declaración inicia con el tipo de dato seguido de la lista de variables separadas por coma, finalizando con punto y coma como todas las instrucciones de C:

```
int x, y, z;
char arreglo[10];
char c, opc='s';
```

En la línea anterior se puede apreciar que es posible inicializar una variable desde la declaración. Únicamente se le debe de asignar un valor a través de una expresión, recordando que la expresión debe regresar un valor del mismo tipo de la variable declarada.

Operadores aritméticos.

Los operadores aritméticos en c son:

Suma	+	
Resta	-	
División	/	
Multiplicación	*	
Módulo	%	No funciona para tipos float o double

Operador de asignación.

El operador de asignación es el símbolo de igual "=", que no se debe de confundir con el operador relacional de igualdad "==".

```
val1=10.5;
val2=11 *(val3=5);
```

Podemos ver dos ejemplo de asignación en las líneas anteriores. Es importante mencionar que el operador de asignación puede ir dentro de una expresión siempre y cuando tenga a su izquierda un dato modificable (como una variable simple, un elemento de arreglo o campo de un registro).

Operadores relacionales.

Para establecer una relación entre dos operandos y devolver un valor de falso o verdadero según corresponda, se usan los siguientes operadores:

==	igualdad
!=	distinto
>	mayor
<	menor

>= mayor o igual
<= menor o igual

Operadores lógicos.

Los operadores lógicos and, or y not se representan en C de la siguiente forma:

And &&
Or ||
Not !

Hay que recordar que en C no existen los valores booleanos: cero se toma como falso y diferente de cero es verdadero.

Operador condicional.

El operador condicional es un operador terciario (tres operandos). En realidad su uso se recomienda para la elaboración de macros, pero es posible utilizarlo en cualquier parte del código. La sintaxis del operador es:

expresión1 ? **expresión2** : **expresión3**

Donde expresión1 se evalúa para obtener un resultado de verdadero o falso. Si **expresión1** es verdadera, se ejecuta **expresión2**; de lo contrario, se ejecuta **expresión3**.

Operador coma.

El operador coma se utiliza para colocar dos expresiones donde sólo se permite una. Se realizan las dos expresiones, pero el valor de la expresión de la izquierda se descarta.

Este operador normalmente se usa en el ciclo for, para incluir más de una variable [4]:

```
for(minimo =0, maximo = longitud; minimo < maximo; minimo++, maximo--)
        <instrucciones>
```

2. Control.

En este apartado se mostraran las estructuras de control de flujo que proporciona C: Estructuras condicionales y estructuras iterativas.

Antes de mencionar las estructuras, es conveniente definir algunas convenciones que se usarán para escribir la sintaxis de cada instrucción:

Expresión	Donde *expresión* es la combinación de constantes, variables, operadores y funciones.
expr_discreta	Es una expresión pero que devuelve un resultado discreto (entero o caracter).
Instrucciones	Puede ser una única instrucción o un bloque de instrucciones encerradas entre llaves { }.

Estructuras condicionales.

C ofrece tres estructuras condicionales o de decisión básicas:

- Decisión simple: if
- Decisión doble: if-else
- Decisión múltiple: switch.

Otra forma de control de flujo es el operador condicional *? :* mencionado en la sección anterior.

Decisión simple (if).

La forma más general de una instrucción condicional if es:

```
if (expresión)
        instrucciones
```

Recordando que en C no existen los valores booleanos, si el resultado de la expresión es diferente de cero las instrucciones de la condición son ejecutadas; de lo contrario -si la expresión es cero- se omite la ejecución de *instrucciones* y ejecuta las instrucciones

Decisión doble (if-else)

La instrucción de decisión doble permite tomar otro camino en caso de que el resultado de la expresión sea cero (falso). La estructura general se muestra a continuación:

```
if (expresión)
        instrucciones1
else
        instrucciones2
```

Decisión múltiple (switch)

La decisión múltiple permite escoger el camino a seguir a partir del resultado de una expresión. Cada camino debe ser un valor constante y por lo tanto no puede ser otra expresión. La sintaxis es la siguiente:

```
switch (expr_discreta){

        case constante1:
                instrucciones1
                [break;]

        case constante2:
                instrucciones2
                [break;]

        . . .
        case constanten:
                instruccionesn
                [break;]
        default:
                instrucciones;
}
```

En la instrucción switch, se evalúa la expresión y toma -si existe- la opción en que el resultado coincida con el valor constante de algún "caso". Si no coincide ninguna opción, se toma el camino por omisión (*default*).

Hay que mencionar otro punto importante. En C se requiere opcionalmente la instrucción break al finalizar las instrucciones de alguna opción. Si no se pusiera esta instrucción, el programa seguiría ejecutando las líneas de código inmediatas, sin saltarse al final del código de la estructura switch. Sin embargo, esta característica puede ser aprovechada por ejemplo para ejecutar un sólo bloque de código para diferentes opciones.

Estructuras iterativas.

En C se incluyen tres estructuras de control, iterativas:

- Estructura iterativa while.
- Estructura iterativa for.
- Estructura iterativa do-while.

Estructura iterativa while.

La forma general de la estructura while es la siguiente:

```
while (expresión)
        instrucciones
```

Esta estructura hace que *instrucciones* se ejecute mientras la evaluación de la expresión regrese un valor diferente de cero - verdadero -, en caso contrario se omite la ejecución de *instrucciones* y continua con las líneas de código que están fuera de la estructura while. La expresión se evalúa al principio, así que es posible que *instrucciones* no se ejecute ni una sola vez, si en la primera evaluación el resultado de expresión es cero.

Ejemplo: Programa que muestra el uso de la estructura repetitiva while

```
#include <stdio.h>
#include <conio.h>
void main()
{
	int contador=1;

	while (contador<=10){
		printf("Número: %d\n", contador);
		getch();
		contador++;
	}
}
```

Estructura iterativa for.

La instrucción iterativa for se describe de forma general a continuación:

for (*expresión1; expresión2; expresión3*)
	instrucciones

Donde generalmente *expresión1* es una inicialización de una variable que se utiliza como contador; *expresión2* una expresión condicional; y, *expresión3* el incremento o decremento de la variable contador. Aunque como hemos visto, C permite muchas libertades en el manejo de las expresiones, por lo que en realidad pueden ser cualquier tipo de expresión permitida en C. Sin embargo hay que tener en cuenta que:

Expresión1. Se realiza únicamente en al inicio de la ejecución de la instrucción for.
Expresión2. Se realiza en cada iteración y se evalúa su resultado: si es cero se sale de la estructura for; diferente de cero permite otra iteración de *instrucciones.*

Esta estructura normalmente se utiliza cuando se conoce el número de iteraciones que se desea realizar.

Ejemplo: Programa que muestra el uso de la estructura repetitiva for

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

void main()
{
	int contador;

	for(contador=1; contador<=10; contador=contador+1){
		printf("Número: %d\n", contador);
		getch();
	}
}
```

Estructura iterativa do-while.

La sintaxis de la estructura iterativa do-while es:

do
instrucciones
while (*expresión*);

En este caso al igual que en la estructura while, se ejecuta *instrucciones* mientras la expresión devuelve un valor diferente de cero -verdadero-, y en caso contrario se sale de la estructura. La diferencia radica en que la evaluación de la expresión se realiza al final de la estructura y no al principio. De esta forma se tiene garantizada al menos una vez la ejecución de *instrucciones.*

Es similar a las estructuras repeat-until de Pascal u otros lenguajes, salvo por la evaluación de la expresión, que por lo general para los otros lenguajes se itera en un repeat hasta que la condición se cumpla, mientras la condición sea falsa.

Ejemplo: Programa que muestra el uso de la estructura repetitiva do-while

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

void main()
{
      int contador=1;

      do {
            printf("Número: %d\n", contador);
            getch();
            contador++;
      }while (contador<=10)
}
```

3. Funciones.

Para resolver un problema complejo es necesario descomponerlo en partes. Esta ha sido una solución natural dadas las limitaciones del hombre de poder atacar todo un problema de gran tamaño al mismo tiempo. La programación no es la excepción, y es normal que un programa este compuesto en realidad de varios subprogramas que realicen cada uno alguna tarea específica.[3]

En C podemos definir una función de la siguiente forma:

```
tipo nombre ( lista_parámetros )    /*encabezado*/
{                                   /*cuerpo de la función*/
        instrucciones
}
```

Donde:

Tipo. Es el tipo de dato que puede devolver la función, el tipo puede ser simple o estructurado. Si no se indica el tipo de dato C asume que es de tipo entero.
Nombre. Se debe indicar un nombre que identifique a la función.
Lista_parámetros. Se incluyen los tipos y nombres de los parámetros que va a manejar la función.
Instrucciones. El código de la función agregando las declaraciones de las variables locales.

En realidad el tipo, la lista de parámetros y las instrucciones son opcionales[1], de forma que podríamos tener una función de esta forma:

```
miFuncioNula( ) { }
```

No se le declara tipo -por lo tanto es de tipo entero-, no recibe parámetros y no tiene código de instrucciones. Obviamente está función no realizaría nada pero es válida para el compilador.

A continuación se muestra una función de ejemplo que realiza el cálculo del factorial:

```
int factorial (int n)
{
        int i, mult;

        mult=0;
        for(i=2: i<=n; i++)
                mult = mult * i;

        return mult;
}
```

Esta función recibe un valor entero a través del parámetro n y calcula el factorial en la variable local mult. Al término de la función el valor del factorial es devuelto a través del nombre de la función -la función es declarada de tipo entero- gracias a la instrucción return.

[3] Matemáticamente una función es una operación que toma uno o más valores llamados argumentos y produce un valor (resultado)

La instrucción return.

Para que la función regrese algún valor es necesario utilizar la instrucción return, esta instrucción es opcional, así que puede o no agregarse[2].

La sintaxis de la instrucción return es:

return [*expresión*];

Donde podemos apreciar como lo denota la sintaxis, que va seguida opcionalmente de una expresión. El resultado de esa expresión es que el regresará la función.

Return provoca además la salida de la ejecución de la función, y no tiene que ir necesariamente al final de la función. Sin embargo, no se recomienda abusar del uso de la instrucción return; por claridad en el código es preferible usar el mínimo de instrucciones de return.

Prototipo de funciones.

Las funciones en C deben de ser declaradas antes de poder ser usadas, de manera que tenemos dos opciones, o agregamos el código de la función antes de usar dicha función, o hacemos uso de lo que ANSI C definió como prototipo de funciones, que no es otra cosa que el encabezado de una función.

De esta manera se cubre la necesidad del compilador de saber la cantidad y tipo de los parámetros y del tipo de la función, aunque el código de la función se encuentre posteriormente, esto nos permite a nosotros una mayor claridad al tener al principio únicamente los prototipos y después de la o las funciones principales ya tendremos entonces la definición completa de las funciones complementarias. En el prototipo no es necesario definir el nombre de los parámetros.

Sintaxis para los prototipos:

tipo nombre(lista_parámetros);

Ejemplo: prototipo de la función de cálculo de factorial:

int factorial (int);

Llamada de funciones.

La llamada a una función es cuando se solicita la ejecución de una función que ya se encuentra definida previamente. En este caso se debe incluir el nombre de la función y entre paréntesis la lista de los datos que se envían como parámetros. Los datos deben coincidir en número y tipo con los declarados en el prototipo y la definición de la función.

Es importante mencionar que en C no existe el paso de parámetros por referencia[2], todos los parámetros se pasan por valor. Sin embargo, es posible por medio de variables de tipo apuntador simular el paso por referencia.

Ejemplo de uso de funciones en C.

Se presenta a continuación un ejemplo de cálculo y desplegado de una tabla de potencias:

```
#include <stdio.h>
#define N 7

/*se definen los prototipos de las funciones*/
long pot(int, int);   /*long int es implícito*/
void imp_encab(void);
void imp_tab_pot(int);

void main(void){

      /*llamadas a las funciones*/
      imp_encab();
      imp_tab_pot(N);
}

/*definiciones de las funciones*/
void imp_encab(void){
      int i;
      printf("\n      -------  Tabla de potencias  --------\n");
      printf("1");
      for(i=2; i<=N; i++)
            printf("%9d", i);
      putchar('\n');
      printf("-------------------------------------------------------\n");
}

void imp_tab_pot(int n){

      int i, j;

      for(i=1; i<=n; i++){
            for(j=1; j<=n; j++)
                  if(j==1)
                        printf("%ld", pot(i,j));
                  else
                        printf("%9ld", pot(i,j));
            putchar('\n');
      }
}

long pot(int m, int n){
      int i;
      long producto=1;

      for(i=1; i<=n; i++)
            producto*=m;
      return producto;
}
```

4. Arreglos.

Existen estructuras de arreglo que pueden ser utilizadas en C. Un arreglo se define como una colección finito, homogénea y ordenada de elementos. Lo que se necesita para definir un arreglo en C es especificar el tipo de datos de los elementos del arreglo, el nombre del arreglo y el tamaño del mismo:

```
Tipo      nombre[tamaño];
```

Ejemplos:

```
char arreglocar[10];      /*arreglo de caracteres de 10 elementos*/
int horas[14];            /*arreglo de enteros de 14 elementos*/
```

Si se requiere un arreglo de más dimensiones únicamente se añade la longitud de las dimensiones extras entre corchetes con el tamaño de la dimensión:

```
int cubo [3][2][4];
```

Inicialización de un arreglo.

Un arreglo puede ser inicializado de diferentes formas [5]:

- Durante la ejecución de un programa, asignando o almacenando los datos de una lectura en el arreglo.
- En el momento de crear el arreglo, indicando los datos que va a contener.
- Por omisión, al ser creados si estos arreglos son globales o estáticos.

Uso de arreglos.

En C, los arreglos pueden ser manipulados de la misma forma que los datos simples:

- Asignación o escritura.
- Lectura o consulta.

Además, los elementos en su conjunto pueden ser ordenados o modificados por medio de algoritmos que resuelven procesos que tiene que ver con el manejo de grupos de datos.

Hay que tener en cuenta algunas características de los arreglos que son particulares a C:

- En C el límite inferior del arreglo es accesado con el índice en cero (arreglo[0]). Por lo tanto si tenemos un arreglo de 10 elementos tenemos que manejar al índice con valores que van del cero al nueve.
- C no hace una comprobación de los límites inferior y superior, permitiendo una ejecución más rápida del código en el manejo de arreglos, pero dejando la responsabilidad al programador. Si no se tiene cuidado es posible acceder por medio del nombre del arreglo y su índice, a celdas de memoria que no correspondan a ese arreglo. Si el error se da en una consulta, lo peor que puede pasar es que el programa arroje resultados incorrectos; pero, si se da en una escritura y se modifica una celda ajena al arreglo, es posible alterar información del programa mismo o de otro programa.

- En C los arreglos unidimensionales se utilizan también para el manejo de cadenas, declarando arreglos de tipo *char*. Hay que considerar un elemento de más al tamaño de la cadena que se piense almacenar para el carácter especial de fin de cadena '\0'.

Ejemplo de operaciones comunes en arreglos:

int x[10];

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]	X[8]	X[9]
14.0	12.0	8.0	7.0	6.41	5.23	6.15	7.25	4.3	1.2

Podemos ver unos ejemplos de operaciones básicas:

Operación	Descripción
x[3]=45	Asignar en la posición 3 del arreglo x el valor de 45.0
suma= x[0]+ x[2]	En la variable suma se almacena el resultado de la suma de x[0] y x[2] (22.0)
suma=suma+x[9]	Acumula en la variable suma el valor de x[9], suma queda con 23.2
x[4]=x[4]+3.5	Suma 3.5 a la celda 5 del arreglo x, el valor para x[4] será 9.91
x[5]=x[0]+x[0+1]	La suma de x[0] y x[1] queda en x[5], el valor en x[5] será de 26
x[10]=1000	Asignación fuera de los límites del arreglo. C no notifica del error. Es responsabilidad del programador evitar que ocurra una situación como esta.

Ejemplo de un programa con uso de arreglos en C:

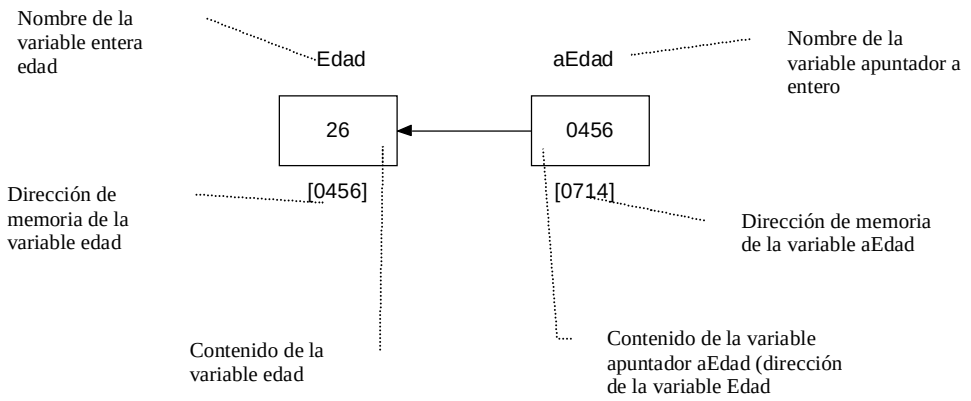
El siguiente programa en C simula la función de lectura de cadena. Declara un arreglo de caracteres de 9 elementos, de manera que la cadena máxima que puede manejar es de 8, reservando el elemento cad[8] -noveno elemento- para el símbolo de fin de cadena (únicamente en el caso de que la cadena ocupe los 8 elementos).

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

void main() {
  char cad[9];
  int i=0;

  clrscr();
  cad[8]='\0';
  do{
      cad[i]=getche();
      if (cad[i]=='\r')
            cad[i]='\0';
  }while(i<7 && cad[i++]!='\r');


  printf("\n%s", cad);
}
```

5. Apuntadores.

Un apuntador es una variable que contiene una dirección de memoria. Esta dirección puede ser la posición de otra variable en la memoria. Por ejemplo:

```
int edad, *aEdad;

Edad=26;
AEdad=&Edad;
```



En el ejemplo tenemos dos variables une de tipo entero llamada Edad y otra de tipo apuntador a entero llamada aEdad. Posteriormente Edad toma el valor de 26 y aEdad toma como valor la dirección de la variable Edad (0456). Podemos decir entonces que aEdad apunta a Edad, y a través de aEdad puedo modificar el valor de la variable Edad.

La declaración de un apuntador como ya vimos en el ejemplo se hace:

```
tipo  *nombre_var;
```

Se agrega * como prefijo al declarar una variable de tipo apuntador. Tipo define el tipo de dato al que va a hacer referencia el apuntador.

Operadores de apuntador.

& Operador de dirección o referencia. Devuelve la dirección de memoria de la variable.
* Operador de indirección o "desreferencia"[4]. Devuelve el valor situado en la dirección del operando. Se dice que da acceso a la variable que señala el apuntador.

Ejemplos. Supongamos un apuntador p y dos variables c y d de tipo char.

```
Char *p, c, d;
p=&c;        Se asigna la dirección de c a la variable apuntador p
d=*p;        Asigna el contenido de c (al que apunta p) a la variable d
```

[4] Desreferencia como muchas otras palabras utilizadas en el ámbito de la computación no existe realmente en el español, pero es utilizado por algunos autores o por la traducción literal de documentos en inglés.

Un pequeño programa de ejemplo. Inicializa una variable entera i con el valor de 100, posteriormente se asigna la dirección de i al apuntador pi. Después la variable val recibe el contenido de lo apuntado por pi (es decir 100) y finalmente se despliega el contenido de val.

```
#include<stdio.h>
void main(){
      int *pi, i, val;

      i=100;

      pi=&i;

      val=*pi;

      printf("%d", val);
}
```

Operadores aritméticos para apuntadores.

Las únicas operaciones que se pueden realizar con variables de apuntador son la suma y la resta, de manera que los operadores válidos son:

+	suma
-	resta
++	incremento
--	decremento

La aritmética de operadores no suma las direcciones de memoria, sino elementos. Esto quiere decir que si yo incremento en uno a una variable de apuntador a entero no se incrementara un byte, sino dos bytes porque es el espacio que ocupa un entero en memoria.[5]

Más ejemplos con apuntadores:

int x=10, y=2, z[14], *p;

p=&x;	p apunta ahora a la variable x
y=*p;	y contiene ahora el valor de 10
*p=0;	x es asignada con un valor de cero a través del apuntador p.
p=&z[2];	p apunta ahora a z[2]
*p=*p+2;	incrementa en dos lo apuntado por p.
++*p	incrementa en uno lo apuntado por p.
(*p)++	incrementa en uno lo apuntado por p. los paréntesis son necesarios para que no incremente a p.

[5] Es lógico que así funcione ya que lo práctico aquí es apuntar al siguiente entero (por ejemplo recorrido de un arreglo).

Apéndice A. Funciones del Lenguaje C más comunes.

Librería stdio.h (C estándar)

- printf. printf(cadena", lista de variables). Despliega una cadena en la salida estándar.
- Scanf. Le información de un stream y la almacena en una o más variables.
- getchar. Es una macro que toma un carácter de la entrada estándar (stdin). Declaración: int getchar(void)
- Putchar. Es una macro que pone un carácter en la salida estándar (stdout). Declaración: int putchar(int c)
- Gets. Toma una cadena de la entrada estándar. Declaración: char *gets(char *s);
- Puts. Coloca una cadena en la salida estándar. Declaración: int puts(const char *s);

Librería conio.h (borland C)

- Cgets, cputs, cprintf, cscanf. El mismo funcionamiento que las funciones de stdio.h pero se envian directamente a la consola.
- Clrscr. Limpia la pantalla. Declaración: void clrscr(void)
- Getch. Recibe un carácter del teclado. No lo despliega en la pantalla. Declaración: int getch(void)
- Getche. Recibe un carácter del teclado. Y lo despliega en la pantalla. Declaración: int getche(void)
- Textbackground. Define un nuevo color para el fondo de la pantalla. Declaración: void textbackground(int newcolor);
- Textcolor. Define un nuevo color para el texto. Declaración: void textcolor(int newcolor);
- Wherex. Devuelve la posición horizontal del cursor. Declaración: int wherex(void);
- Wherey. Devuelve la posición vertical del cursor. Declaración: int wherey(void);

Librería dos.h (borlandc)

- Delay. Retarda la ejecución de un programa en milisegundos. Declaración: void delay(unsigned milliseconds);
- Sleep. Retarda la ejecución de un programa en segundos. Declaración: void sleep(unsigned seconds);

Librería math.h

- Pow. Eleva un número x a una potencia y. Declaración: double pow(double x, double y);
- Sqrt. Raíz cuadrada de un número. Declaración: double sqrt(double x);
- Randomize. Inicializa la generación de números random o aleatorios. Declaración: void randomize(void);
- Random. Devuelve un número aleatorio entre 0 y un numero-1. Declaración: int random(int num);
- Funciones de conversión de tipos de datos: atoi, atol, atof, ltoa, itoa,

Librería string.h (funciones de manejo de cadenas).

- Strcpy. Copia una cadena a otra. Declaración: char *strcpy(char *dest, const char *src);
- Strlen. Calcula la longitud de una cadena. Declaración: size_t strlen(const char *s);
- Strlwr. Convierte una cadena a minúsculas. Declaración: char *strlwr(char *s);
- Strcat. Concatena a la primera cadena la segunda. Declaración: char *strcat(char *dest, const char *src);
- Strcmp. Compara dos cadenas. Devuelve < 0 si s1 < s2; == 0 si s1 == s2; > 0 si s1 > s2.Declaración: int strcmp(const char *s1, const char*s2);

6. Bibliografía.

[1] Kernighan, Brian; Ritchie, Dennis. **El Lenguaje de Programación C.** 2ª edición. Edit. Prentice Hall. 1991.

[2] Keller, AL; Pohl, Ira. **A Book on C.** 3ª edición. Edit. Benjamin Cumnings. 1995.

[3] Pratt, Terrence; Zelkowitz, Marvin. **Programming Languages.** 3ª edición. Edit.Prentice Hall. 1996.

[4] Pappas, Chris; Murray, Willian. **Manual de Borland C++**. 1ª edición. Mc Graw Hill. 1993.