FUNCIONES

Una función es un conjunto de sentencias que se pueden llamar desde cualquier parte de un programa.

Cada función realiza una determinada **tarea** y cuando se ejecuta *return* o termina el código de la función, se retorna al punto en que fue llamada por el programa o función principal.

Como vimos, cada programa C tiene al menos una función main( ); sin embargo, cada programa C puede constar de muchas funciones en lugar de una función main( ) “grande”.

ESTRUCTURA DE UNA FUNCIÓN

tipo_de_retorno nombreFunción (*lista De Parámetros*)

{

cuerpo de la función

return expresión

}

- tipo_de_retorno: Tipo de dato del valor devuelto por la función o la palabra reservada void si la función no devuelve ningún valor.
- nombreFunción: Identificador o nombre de la función.
- lista de parámetros: Lista de parámetros con los tipos, se utiliza el formato siguiente: tipo1 parámetro1, tipo2 parámetro2, ...
- cuerpo de la función: compuesto por sentencias terminadas en punto y coma (;). Se encierra entre llaves de apertura ({) y cierre (}).

EJEMPLO

```
float sumar(float num1, float num2)
{
    float respuesta;
    respuesta = num1 + num2;
    return respuesta;
}
```

PROTOTIPO DE UNA FUNCIÓN

Es la declaración de una función.

Consta de los siguientes elementos: tipo de retorno, nombre de la función, lista de argumentos encerrados entre paréntesis y un punto y coma.

En C no es necesario incluir el prototipo aunque si es recomendable para que el compilador pueda hacer chequeos en las llamadas a las funciones.

Se sitúan normalmente al principio de un programa, antes de la definición de la función main().

EJEMPLO

```
float sumar(float num1, float num2);
```

LLAMADA A UNA FUNCIÓN

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

float sumar(float num1, float num2); //prototipo de la función sumar

int main()
{
    float numero1, numero2, resultado;

    printf("Ingresa primer numero:\n");
    scanf("%f", &numero1);
    printf("Ingresa segundo numero:\n");
    scanf("%f", &numero2);

    resultado = sumar(numero1, numero2);//llamada a la función sumar
    printf("La suma es: %f\n", resultado);

    system("pause");
    return 0;
}

//función sumar
float sumar(float num1, float num2)
{
    float respuesta;
    respuesta = num1 + num2;
    return respuesta;
}
```

EJERCICIOS

Retomamos dos ejercicios de la Práctica 2, para reescribirlos pensando en funciones.

1) Pedir al usuario que ingrese dos números. Luego presentar el siguiente menú:
   1. Informar su suma
   2. Informar su resta
   3. Informar su multiplicación
   4. Informar su división
   5. Salir
   Seleccione una operación:
   Mostrar el resultado de la operación seleccionada.

2) Escriba un programa que pida ingresar un número y a continuación escriba en la consola si el mismo es par o impar.

PARÁMETROS DE LAS FUNCIONES

Los **parámetros** de una **función** son los valores que esta recibe por parte del código que la llama.

Una función puede tener cualquier cantidad de parámetros (incluso cero), los cuales se declaran dentro de los paréntesis con la misma sintaxis de las declaraciones de las variables.

Existen dos formas de pasar parámetros a una función:

- Parámetros por valor
- Parámetros por referencia

PARÁMETROS POR VALOR

C siempre utiliza el método de parámetros por valor para pasar variables a funciones.

Los parámetros por valor reciben copias de los valores de los argumentos que se les pasan. La asignación a parámetros por valor de una función nunca cambian el valor del argumento original pasado a los parámetros.

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

float sumar(float num1, float num2); //prototipo de la función sumar

int main()
{
    float numero1 = 6, numero2 = 8, resultado;
    //    valor          6         8
    resultado = sumar(numero1, numero2); // llamada a la función sumar(6, 8) y
    // asignación del valor devuelto a la variable resultado
    //luego de la llamada a la función los valores de numero1 y numero2 siguen siendo 6 y 8
    printf("La suma es: %f\n", resultado); // 14

    system("pause");
    return 0;
}

float sumar(float num1, float num2)
{   // la variable num1 se inicializa con el valor 6
    // la variable num2 se inicializa con el valor 8
    float respuesta;
    respuesta = num1 + num2;
    // se asigna a la variable respuesta el valor 14 (resultado de 6 + 8)
    num1 = 3;
    num2 = 5;

    return respuesta; //se devuelve el valor 14
}
```

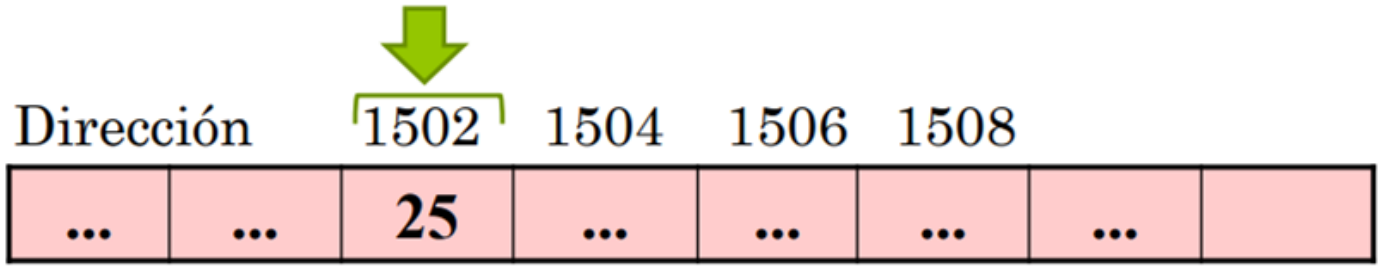
PUNTEROS

Todas las variables vistas hasta este momento contienen valores de datos; por el contrario un puntero es **una variable que contiene una dirección de memoria.**

La declaración de una variable puntero debe indicar el tipo de dato al que apunta.

Una dirección de memoria y su contenido no es lo mismo.

```
int x = 25;
```




```
La dirección de la variable x es 1502
El contenido de la variable x es 25
```

PUNTEROS

Definición de una variable puntero

tipo_de_dato *nombre_variable;

Cada puntero debe llevar su nombre precedido por *

Operador	Propósito
&	Obtiene la dirección de una variable.
*	Define una variable como puntero.
*	Obtiene el contenido de una variable puntero.

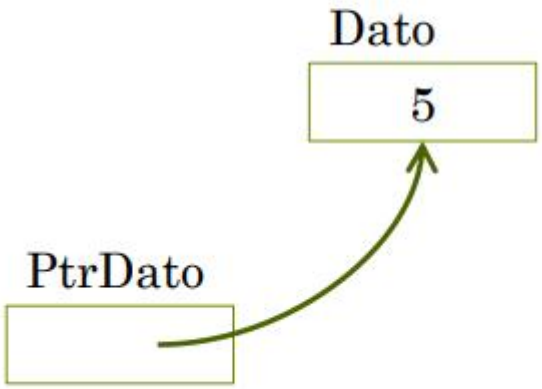
OPERADORES DE PUNTEROS

& operador dirección: devuelve la dirección de la variable a la cual se aplica.

- **Ejemplo**

```
int Dato = 5;
int *PtrDato;

PtrDato = &Dato;
```



OPERADORES DE PUNTEROS

*** operador de indirección:** retorna el valor del objeto hacia el cual apunta su operando.

- **Ejemplo**

```
int Dato = 5, *PtrDato;

PtrDato = &Dato;

printf("%d\n", *PtrDato);
```

PARÁMETROS POR REFERENCIA

Los parámetros por referencia (declarados con *, punteros) reciben la dirección de los argumentos pasados; a éstos les debe de preceder del operador &, excepto los arrays. En una función, las asignaciones a parámetros referencia (punteros) cambian los valores de los argumentos originales.

PARÁMETROS POR REFERENCIA

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

void intercambio(int* a, int* b);  //prototipo de la función intercambio

int main()
{
    int i = 3, j = 50;

    printf("i = %d y j = %d \n", i, j);

    intercambio(&i, &j); //se pasan a la función intercambio() las direcciones de las
 variables i y j

    printf("i = %d y j = %d \n", i, j);

    system("pause");
    return 0;
}

void intercambio(int* a, int* b)
{
    int aux = *a; //asigna a la variable aux el valor de la variable a la que apunta
                  a (es decir, la variable i)
    *a = *b; //asigna el valor de variable que a la que apunta b, a la variable que a
              punta a
    *b = aux; //asigna el valor de aux a la variable que apunta b (es decir,
               la variable j)
}
```

PARÁMETROS DE TIPO ARRAYS

Para C, un array es un puntero que apunta a una zona de memoria reservada en tiempo de compilación.

El nombre de un array es un puntero al inicio del mismo.

Lo arreglos en C se pasan a una función como referencia y no como valor. Esto significa que las modificaciones que hagamos dentro de la función al arreglo que recibimos como parámetro, realmente se realizan en el arreglo original que se utilizó como argumento al momento de llamar a la función.

PARÁMETROS DE TIPO ARRAYS

```c
#include <stdio.h>
#include <stdlib.h>
void reemplazar(char cadena[], int tam);
int main()
{
    char palabra[10];
    printf("Escriba una palabra\n");
    scanf("%s", palabra);

    printf("Palabra ingresada: %s.\n", palabra);

    reemplazar(palabra, 10); //cuando paso un array cómo parametro no hace falta el opera
dor &

    printf("Palabra modificada: %s.\n", palabra);

    system("pause");
    return 0;
}

void reemplazar(char cadena[], int tam)
{   // El tamaño de cadena va a depender del array que le pasen cómo parámetro
    // Para que accedamos a posiciones válidas del array, en general
    // se pasa también cómo parámetro el tamaño del mismo..
   for(int i = 0; i < tam; i++)
    {
        if(cadena[i] == 'a')
        {
            cadena[i] = '8';
        }
    }
}
```

VARIABLES GLOBALES Y LOCALES

Una **variable local** es aquella cuyo ámbito se restringe a la función en la que se declaro.

Una **variable global** se define fuera del cuerpo de cualquier función.

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

float sumar(float num1);

int numero2; //variable global, visible en todas las funciones definidas

int main()
{
    float numero1 = 6, resultado; //variables locales, sólo visibles dentro de esta función
    numero2 = 8; //numero2 es visible ya que es global

    resultado = sumar(numero1);
    printf("La suma es: %f\n", resultado);

    system("pause");
    return 0;
}

float sumar(float num1)
{
    float respuesta; //num1 y respuesta son variables locales
    respuesta = num1 + numero2; //numero2 es visible ya que es global

    return respuesta;
}
```

EJERCICIO

Retomamos el ejercicio complementario del cajero automático y modificamos el código para que cumpla con lo siguiente:

-Contemplar que haya más de una cuenta de cliente, por ejemplo 5 cuentas..

Cuenta 1 – Pass: 234
Cuenta 2 – Pass: 5286
Cuenta 3 – Pass: 945
etc..

-Que la validación de nro. de cuenta y contraseña válidos sea a través de una función llamada login.

- Realizar las operaciones (deposito, extracción, etc) a través de funciones que tengan asignada cada tarea.