Funciones

(Lenguaje C)

El lenguaje C y las funciones

- El lenguaje C es un lenguaje basado en el concepto de funciones.
- Un programa en lenguaje C es una colección de una o más funciones.
- Todo programa en C se basa en la función “main”, que es la función principal del programa, y que se reserva para el inicio de la ejecución de cualquier programa.
- **Una función es una parte del programa compuesta de sentencias o instrucciones, para lograr un objetivo especifico**

Concepto de función
Programación modular

- Los programas resuelven problemas de forma computacional.
- Cuando la complejidad de los problemas aumenta, aumenta la complejidad de los programas.
- Para resolver un problema complejo, se puede adoptar el concepto “divide y vencerás”.
- **Un programa complejo se puede dividir en partes pequeñas de menor complejidad, cada una de ellas con un propósito único e identificable.**
- **Cada una de estas partes se puede programar de forma independiente (módulos - en C se llaman funciones).**
- La función “main”, es la función que se ejecuta apenas el programa inicia la ejecución, y contiene el código que se ejecuta en primer lugar.
- Dentro del “main” habrá llamadas a funciones ya creadas por el propio programador ó de bibliotecas de C (una biblioteca o librería es una colección de funciones).

Concepto de función
Ventajas del uso de funciones

- En una sola función se pueden incluir instrucciones que son ejecutadas repetidamente por distintas partes de un programa.

- Se pueden ejecutar las mismas instrucciones muchas veces, pero pasando distintos datos de entrada para cada ejecución particular.

- Se evita la necesidad de repetir las mismas instrucciones de forma redundante.

- Claridad lógica, **al dividir el problema en partes pequeñas con un objetivo especifico, el mismo es mas fácil de escribir, entender y depurar.**

- Permite al programador escribir una biblioteca a medida, con rutinas de uso frecuente.

Definición de la función

La definición de una función consta de dos partes principales:

especificador de tipo nombre_de_la_función(declaraciones de parámetros)

{

cuerpo de la función

}

- El especificador de tipo determina **el tipo de valor que devuelve** la función. En el caso que no devuelva nada se puede especificar “void” ó simplemente nada.
- El **nombre de la función** es cualquier identificador válido en C, a través del nombre se referencia/invoca/llama a la función.
- La **declaración de parámetros** especifica los tipos y argumentos a recibir por la funciòn separados por coma. **A través de los paràmetros la funciòn recibe informaciòn del programa principal.**
- El **cuerpo** de la función se delimita a través del uso de {}.

Definición de la función
Parámetros ó argumentos

especificador de tipo nombre_de_la_función(**declaraciones de parámetros**)
{
cuerpo de la función
}

- La declaración de parámetros **especifica los tipos y argumentos a recibir** separados por coma. Estos argumentos se denominan argumentos formales, parámetros ó **parámetros formales, son los nombres de los elementos que se transfieren a la función** desde la parte del programa que hace la llamada. Por el contrario**, los argumentos correspondientes a la referencia o llamada a la función se denominan argumentos reales o parámetros reales**, ya que definen la información que realmente se transfiere. El uso es opcional puede no recibir parámetros.

- Los identificadores utilizados como **argumentos formales son "locales" a la función**, no son reconocidos fuera de la función, no tienen porque coincidir con los parámetros reales en cuanto a nombre, pero si deben hacerlo en cuanto a tipo.

Definición de la función
Cuerpo de la función

especificador de tipo nombre_de_la_función(declaraciones de parámetros)
{ **cuerpo de la función**
}

- El cuerpo de la función **es un conjunto de instrucciones que define lo que esta hace**.
- Para **retornar un valor la función lo hace mediante la instrucción return <expresión>.** En la instrucción return se puede incluir solo una expresión.
- La instrucción return hace que se devuelva el control al punto de llamada.
- Si la función no retorna ningún valor, en el especificador de tipo se utiliza void, y en el cuerpo de la función no se utiliza el return, o simplemente se utiliza return solo.
- **El tipo de la expresión return debe ser del mismo tipo que el especificado como valor devuelto en la primera línea de la definición de la función**, sino el compilador intentará convertir automáticamente esta expresión al tipo de datos especificado como devolución.

Definición de la función
Ejemplos

```
long int factorial(int n)
{
int i;
long int prod = 1;
if (n > 1)
for (i = 2; i <= n; ++i)
prod *= i;
return(prod) ;
}

/* calcular el factorial de n */
```

Definición de la función
Ejemplos

```
float precio(float base, float impuesto) /* definición */
{
    float calculo;
    calculo = base + ((base * impuesto)/100);
    return calculo;
}

/*calcula un precio*/
```

Acceso a una función

- Para llamar a una función desde el programa principal (es a su vez una función), **se especifica su nombre, seguido de la lista de argumentos reales encerrados entre paréntesis** y separados por comas, si la función no requiere parámetros entonces solo se utilizaran los paréntesis vacios.
- La llamada puede ser parte de una expresión simple o parte de una expresión compleja.
- Cantidad y tipo: en una llamada a una función habrá un argumento ó parámetro real por cada argumento o parámetro formal. **Cada argumento real debe ser del mismo tipo que los tipos de los argumentos formales.**
- Los argumentos reales pueden ser constantes, variables ó expresiones mas complejas.
- El valor de **cada argumento real es transferido a la función y asignado al correspondiente argumento formal.**
- Cuando se llama a una función dentro de una expresión, el control del programa se pasa a ésta y sólo regresa a la siguiente expresión de la que ha realizado la llamada.

Acceso a una función
Ejemplos

```
#include <stdio.h>
float precio(float base, float impuesto) /* definición */
{
    float calculo;
    calculo = base + ((base * impuesto)/100);
    return calculo;
}
main()
{
    float importe, tasa;
    printf("Ingrese el porcentaje del IVA: ");
    scanf("%f",&tasa);
    printf("\n\nIngrese el precio sin IVA: ");
    scanf("%f",&importe);
    printf("\n\nEl precio a pagar es: %.2f\n", precio(importe, tasa));
    return 0;
}
```

Acceso a una función
Ejemplos

- Vamos a acceder a las funciones primera y segunda desde la función main.

```
#include <stdio.h>
void primera(void)
{
	printf("Llamada a la función primera\n");
	return;
}
void segunda(void)
{
	printf("Llamada a la función segunda\n");
	return;
}
main()
{
	printf("La primera función llamada, main\n");
	primera();
	segunda();
	printf("Final de la función main\n");
	return 0;
	}
```

La salida es:
La primera función llamada, main
Llamada a la función primera
Llamada a la función segunda
Final de la función main

Prototipos

- En los programas vistos, la definición de la función precede al main. De este modo, la definición de la función precede a la primer llamada a la función. También **existe otro enfoque descendente en el que la idea es primero colocar a la función main y luego de esta a la definición del resto de las funciones**. De este modo, la llamada a la función precedería a la definición, por lo que para evitar confusiones al compilador, se utilizan los prototipos.
- Aunque no es obligatorio en C, se pueden utilizar los prototipos que hacen que el compilador desde el primer momento conozca sobre la existencia de la función.
- Estos se declaran luego de los include.
- **La declaración de una función se conoce también como prototipo de la función. En el prototipo de una función se tienen que especificar los parámetros de la función, así como el tipo de dato que devuelve.**

  tipo_de_retorno nombre_de_la_función(lista_de_parámetros);

- **En el prototipo de una función no se especifican las sentencias que forman parte de la misma, sino sus características**. En el mismo no hace falta declarar los nombres de los argumentos (aquí se llaman argumentos ficticios), pero si es necesario declarar su tipo.

Prototipos
Ejemplos

```
#include <stdio.h>
/*prototipo de la funcion*/
int cubo(int base);
main()
{
	int numero;
	for(numero=1; numero<=5; numero++)
	{
		printf("El cubo del número %d es %d\n", numero, cubo(numero));
	}
	return 0;
}
/*definicion de la funcion*/
int cubo(int base)
{
int potencia;
potencia = base * base * base;
return potencia;
}
```

Pasaje de parámetros
Por copia

- Cuando se pasa un valor a una función mediante un argumento real, **se copia el valor del argumento real a la función.**

- En otras palabras, **se pasa una copia del valor del argumento real y no el argumento en sí** (por ello, este procedimiento se conoce en algunas ocasiones como **paso por copia ó por valor).**

- **Al pasar** una copia del argumento original a la función, **cualquier modificación que se realice sobre esta copia no tendrá efecto sobre el argumento original utilizado en la llamada de la función.**

Pasaje de parámetros
Ejemplo de pasaje por copia

```
#include <stdio.h>
void modificar(int variable);
main()
{
	int i = 1;
	printf("\ni=%d antes de llamar a la función modificar", i);
	modificar(i);
	printf("\ni=%d después de llamar a la función modificar", i);
}
void modificar(int variable)
{
	printf("\nvariable = %d dentro de modificar", variable);
	variable = 9;
	printf("\nvariable = %d dentro de modificar", variable);
}
```

La salida es la siguiente:

- i=1 antes de llamar a la función modificar
- variable = 1 dentro de modificar
- variable = 9 dentro de modificar
- i=1 después de llamar a la función modificar

LOS RESULTADOS MUESTRAN QUE NO SE MODIFICAN LOS VALORES DE LOS ARGUMENTOS REALES

Pasaje de parámetros
Por referencia

- Pasar argumentos por valor ó por copia, tiene sus ventajas e inconvenientes. Una ventaja es que se puede proporcionar una expresión en lugar de una variable. Si el argumento real es una variable, **se lo protege de modificaciones dentro de la función**. LA TRANSFERENCIA DE INFORMACIÓN ES EN UN SOLO SENTIDO.

- Sin embargo, **en muchas ocasiones lo que queremos es que una función cambie los valores de los argumentos que le pasamos. Para lograrlo se utiliza lo que se conoce como paso de argumentos por referencia. En estos casos, no se pasa una copia del argumento, sino el argumento mismo.**

- Cuando realizamos un paso de argumentos por referencia en C, **realmente lo que estamos pasando son direcciones de memoria**, es decir un "apuntador" al lugar donde está realmente el dato guardado. Esto permite que los datos utilizados en la llamada a la función puedan ser accedidos y modificados por esta.

- En los pasajes de parámetros por referencia, **los argumentos reales que se utilizan en las llamadas a funciones son direcciones de memoria** (punteros). (*)

(*) EL TEMA PUNTEROS SE EXPLORA A DETALLE EN LA UNIDAD CORRESPONDIENTE.

Pasaje de parámetros
Por referencia

- Se debe especificar en la “definición” de la función, **a través del símbolo * seguido al tipo de argumento del que estamos pasando su dirección** .

- **En la llamada a la función, para indicar que estamos pasando la dirección de memoria del dato a pasar, se debe anteponer al mismo el símbolo &** (caso especial de arrays no hace falta). Al finalizar la ejecuciòn de la función, el valor de dicha dirección permanece igual y lo que se ha modificado es el contenido de esa dirección de memoria.

- Dentro de la función se utilizan los punteros para trabajar con las direcciones de memoria *(*variable).*

(*)EL TEMA PASAJES POR REFERENCIA EN FUNCIONES SE EXPLORA TAMBIEN A DETALLE EN LAS UNIDADES CORRESPONDIENTES A ARRAYS Y PUNTEROS.

Pasaje de parámetros
Ejemplo de pasaje por referencia

```
#include <stdio.h>
void modificar(int *variable);
main()
{
	int i = 1;
	printf("\ni=%d antes de llamar a la función modificar", i);
	modificar(&i);
	printf("\ni=%d después de llamar a la función modificar", i);
	return 0;
}
void modificar(int *variable)
{
printf("\nvariable = %d dentro de modificar", *variable);
*variable = 9;
printf("\nvariable = %d dentro de modificar", *variable);
}La salida de este ejemplo sería:
```

- i=1 antes de llamar a la función modificar
- variable = 1 dentro de modificar
- variable = 9 dentro de modificar
- i=9 después de llamar a la función modificar

LOS RESULTADOS MUESTRAN QUE SE MODIFICAN LOS VALORES DE LOS ARGUMENTOS REALES

Pasaje de parámetros
Transferencias de inf. - variables globales

```
#include <stdio.h>
void total (int);
void imprima ();
int acum;
int main()
{
        int cont;
        acum=0;
        for(cont=0;cont<10;cont++)
        {
                total(cont);
                imprima();
        }
}

void total(int x)
{
        acum=acum+x;
}

void imprima ()
{
        int cont;
        for(cont=0;cont<5;cont++)
                printf("*");
        printf("La suma es: %d \n",acum);
}
```

- La variable acum es global.
- La función total no puede acceder a la variable cont local del main.
- La variable cont de la función imprima no tiene nada que ver con la variable cont del main

Recursividad

- Una función puede llamar a otras funciones ó a si misma.
- **La recursividad es un proceso mediante el cual una función se llama a si misma de forma repetida hasta alguna condición determinada.**
- El proceso se utiliza para cálculos repetitivos.
- Para que un problema se pueda resolver recursivamente se deben dar dos condiciones: **el problema se debe escribir de forma recursiva y la especificación del problema debe tener una condición de fin.**

Recursividad

- Ejemplo: calculo del factorial:

Forma de expresarlo A:

n!=1*2*3*4* ….*n

Forma de expresarlo B (forma recursiva):

n! = n * (n-1)!

El factorial de un numero se expresa como el numero por el factorial del numero anterior.

A su vez, esta forma anterior de expresarlo, tiene la condición de fin 1!=1.para el proceso recursivo.

Recursividad
Ejemplo

```
#include <stdio.h>
long int factorial (int n); /* prototipo de función */
main ()
{
int n;
printf ("n =  "); /* leer la cantidad entera */
scanf ("%d", &n);
printf( "n! = %ld\n", factorial (n)) ; /* calcular y visualizar el factorial */
}
long int factorial(int n)
{
/* calcular el factorial */
if (n <= 1)
return (1) ;
else
return(n * factorial(n - 1));
}
```

Recursividad

- La función factorial se llama a sí misma recursivamente, con un argumento real (n - 1) que decrece en cada llamada sucesiva. Las llamadas recursivas terminan cuando el valor del argumento real se hace igual a 1.
- Con If – Else se incluye una condición de terminación que se cumple cuando el valor de n es menor o igual a 1.
- Cuando se ejecuta el programa, se accede a la función factorial repetidamente, una vez en main() y (n - 1) veces. desde dentro de la funciòn misma.
- **Cuando se ejecuta un programa recursivo, las llamadas recursivas no se ejecutan inmediatamente. Lo que se hace es colocarlas en una pila hasta que se encuentra la condición de terminación de la recursividad. Luego se ejecutan las llamadas a la función en orden inverso a como se generaron, como si se fueran «sacando» de la pila.**
- El orden inverso de ejecución es una característica de todas las funciones recursivas. Si una función recursiva contiene variables locales, se creará un conjunto diferente de variables locales durante cada llamada. Los nombres de las variables locales serán, siempre los mismos, sin embargo, las variables representarán un conjunto diferente de valores cada vez que se ejecute la función. Cada conjunto de valores se almacenará en la pila; así se podrá disponer de ellas cuando el proceso recursivo se «deshaga», es decir cuando las llamadas a la función se «saquen» de la pila y se ejecuten.

Recursividad

- Por ejemplo para la ejecución de 4!, se vería algo así:

  Llamadas a la función desde el main:

  factorial(4)

  Llamadas a la función desde la función misma:

  factorial(4) = return (4 * factorial(4-1)) = return(4 * factorial(3))

  factorial(3) = return(3 * factorial(3-1)) = return(3 * factorial(2))

  factorial(2) = return(2 * factorial(2-1)) = return(2 * factorial(1))

  Devolución de los valores (cuando llega a la clausula de final):

  factorial (1)= return(1)

  factorial(2) = 2 * factorial(1) = 2 * 1 = 2

  factorial(3) = 3 * factorial(2) = 3 * 2 = 6

  factorial(4) = 4 * factorial(3) = 4 * 6 = 24

Bibliotecas y librerías

- Muchas veces las funciones se almacenan en archivos externos, por lo cual se hace necesario incluir en nuestro programa sentencias que indiquen donde están esas funciones. Esto se realiza mediante las sentencias “include”.

- La sentencia include es una directiva de Preprocesador (una instrucción para el compilador que le indica donde estarán las funciones que utilizamos)

Principal material bibliográfico utilizado

- Fundamentos de programación C/C++ - 4ta Edición – Ernesto Peñaloza Romero – Alfaomega
- Programación en C - 2da Edición – Byron S. Gottfried – Mac Graw Hill
- www.jorgesanchez.net
- Lenguaje C – Apunte de los profesores Adolfo Beltramo y Nélida Matas.

FIN