Universidad Tecnológica Nacional
Facultad Regional Resistencia
Técnico Universitario en Programación

PROGRAMACIÓN I

Funciones

Concepto de Función

Una FUNCIÓN es un subprograma que realiza una tarea determinada y bien acotada a la cual le pasamos datos y nos devuelve datos. Esta función se ejecuta cuando se lo llama (llamada a la función)

Ventajas:

- El programa es mas simple de comprender ya que cada módulo se dedica a realizar una tarea en particular.
- La depuración queda acotada a cada módulo.
- Las modificaciones al programa se reducen a modificar determinados módulos.
- Cuando cada módulo esta bien probado se lo puede usar las veces que sea necesario sin volver a revisarlo.
- Se obtiene una independencia del código en cada módulo.

Modularidad – Cohesión y Acoplamiento

Modularidad

- **Para resolver un problema complejo de desarrollo de software, conviene separarlo en partes más pequeñas, que se puedan diseñar, desarrollar, probar y modificar, de manera sencilla y lo más independientemente posible del resto de la aplicación.**
- **Esas partes, cuando se quiere usar un nombre genérico, habitualmente se denominan módulos.**
- **En programación estructurada se modulariza la solución, el "cómo" del desarrollo.**

Modularidad – Cohesión y Acoplamiento

Cohesión

La cohesión se refiere a que cada módulo del sistema se refiera a un único proceso o entidad.

- **A mayor cohesión, mejor: el módulo en cuestión será más sencillo de diseñar, programar, probar y mantener.**

- **En el diseño estructurado, se logra alta cohesión cuando cada módulo (función o procedimiento) realiza una única tarea trabajando sobre una sola estructura de datos.**

- **Un test que se suele hacer a los módulos funcionales para ver si son cohesivos es analizar que puedan describirse con una oración simple, con un solo verbo activo. Si hay más de un verbo activo en la descripción del procedimiento o función, deberíamos analizar su partición en más de un módulo, y volver a hacer el test.**

Modularidad – Cohesión y Acoplamiento

Acoplamiento

El acoplamiento mide el grado de relacionamiento de un módulo con los demás.

A menor acoplamiento, mejor: el módulo en cuestión será más sencillo de diseñar, programar, probar y mantener.

En el diseño estructurado, se logra bajo acoplamiento reduciendo las interacciones entre procedimientos y funciones, reduciendo la cantidad y complejidad de los parámetros y disminuyendo al mínimo los parámetros por referencia y los efectos colaterales

Características de las funciones en C:

- Una de estas funciones tiene que llamarse main.
- La ejecución del programa siempre comenzará por las instrucciones contenidas en main.
- Se pueden subordinar funciones adicionales a main, y posiblemente unas a otras.
- Si un programa contiene varias funciones, sus definiciones pueden aparecer en cualquier orden, pero deben ser independientes unas de otras. Esto es, una definición de una función no puede estar incluida en otra.
- Cuando se accede a una función desde alguna determinada parte del programa (cuando se «llama» a una función), se ejecutan las instrucciones de que consta. Se puede acceder a una misma función desde varios lugares distintos del programa.

Características:

- Una vez que se ha completado la ejecución de una función, se devuelve el control al punto desde el que se accedió a ella. Generalmente, una función procesará la información que le es pasada desde el punto del programa en donde se accede a ella y devolverá un solo valor.

- La información se le pasa a la función mediante unos identificadores especiales llamados argumentos (también denominados parámetros) y es devuelta por medio de la instrucción return.

- Sin embargo, algunas funciones aceptan información pero no devuelven nada (por ejemplo, la función de biblioteca printf), mientras que otras funciones (la función de biblioteca s canf) devuelven varios valores.

Tipos de Funciones

Funciones Predefinidas:

- Están definidas en el lenguaje de programación y pueden ser utilizadas directamente. Ej. POW

Ejemplo: la función strlen, que a partir de una cadena de caracteres que recibe como parámetro de entrada calcula un valor, que es la longitud de esa cadena.

Funciones definidas por el usuario :

•Utilizadas cuando las funciones predefinidas no permiten realizar el tipo de cálculo deseado, y es el usuario el que debe implementar, mediante estructuras de control adecuadas, la tarea a realizar.

PROCEDIMIENTO

En el lenguaje C no se habla habitualmente de procedimientos, sino sólo de funciones. Pero existen de las dos tipos.

Procedimientos serían, por ejemplo, la función printf no se invoca para calcular valores nuevos, sino para realizar una tarea sobre las variables.

En C se usa una función de tipo void para realizar un procedimiento.

Una función tipo void no necesariamente tendrá la sentencia return. Si una función de tipo void hace uso de sentencias return, entonces en ningún caso debe seguir a esa palabra valor alguno: si así fuera, el compilador detectará un error y no compilará el programa.

Return

- **Si la función es de un tipo de dato distinto de void, entonces en el bloque de la función debe escribirse, al menos, una sentencia return. A continuación de la palabra return, deberá ir el valor que devuelve la función:, siempre del mismo tipo al tipo de la función .**

Fuerza la salida inmediata del cuerpo de la función y se vuelve a la siguiente sentencia después de la llamada.

La forma general de la sentencia return es:

return [expresión];

Si el tipo de dato de la expresión del return no coincide con el tipo de la función entonces, de forma automática, el tipo de dato de la expresión se convierte en el tipo de dato de la función.

Variables Globales y Locales

Todas las variables que se encuentren definidas dentro de las llaves de una función (recuerde que main también es una función) tienen validez dentro de dicha función y se llaman VARIABLES LOCALES.

Ahora si una variable puede ser usada desde cualquier función y durante el transcurso de todo el programa, esa es una VARIABLE GLOBAL. Las variables globales de definen fuera de cualquier función , normalmente debajo de los *include* que se colocan al comienzo del programa.

Pasaje de Parámetros por Valor y por Referencia

Se llama pasaje por valor cuando a la función se le pasa como parámetro actual el valor de la variable.

Se llama pasaje por referencia cuando a la función se le pasa como parámetro actual la dirección de memoria de una variable. Tenga en cuenta que en este caso la variable que recibe deber ser un puntero , ya que lo que se esta pasando es una dirección de memoria.

Declaración de una Función

Tipo_devuelto	Nombre_de_funcion (*tipo* variable_1, *tipo* variable_2 , ... , *tipo* variable_N)

Tipo_devuelto es el tipo de dato que devuelve la función luego de terminada su ejecución

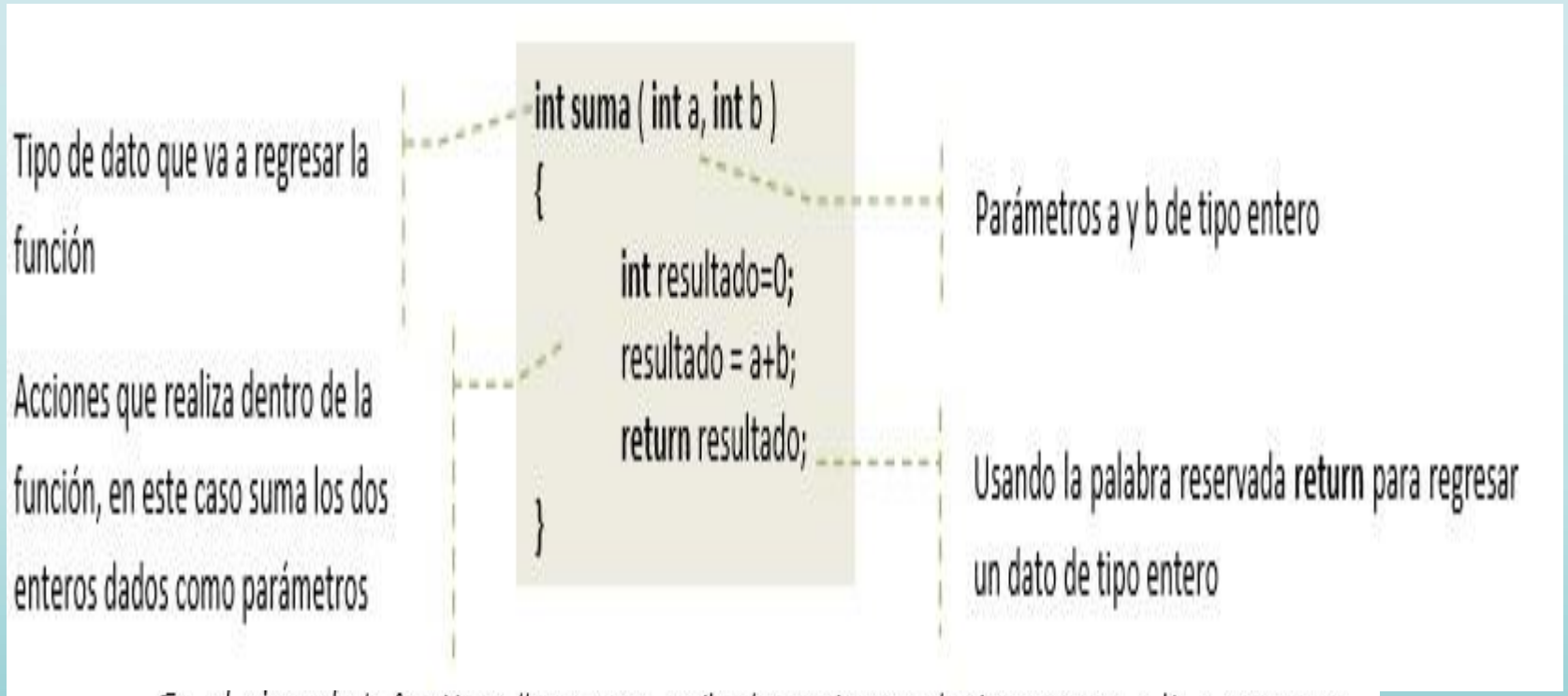
Nombre_de_función es el nombre de la función

tipo variable_1 es el tipo y nombre de la variable que se le pasa a la función.

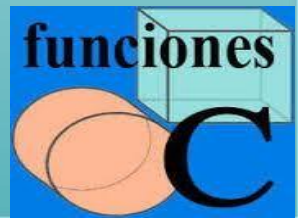
Según la declaración de la función, se devuelve solamente un valor.

```
<tipo de retorno> <nombre de la función> ( <lista de parámetros> )
{
        <acciones que realiza la función>
}
```

Declaración de una Función - Ejemplo



En el ejemplo la función se llama suma, recibe dos parámetros de tipo entero (a y b), y retorna un entero (que es la suma de los parámetros). Una función siempre retorna un solo valor (en casos eventuales no podría no devolver nada si se usa ***void*** como tipo de retorno). Si fuese necesario de devolver más de un valor, debería realizarse por medio de los parámetros, que dejarían de ser sólo de entrada, para convertirse en parámetros de entrada y salida.



Programa sin función

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

int main()
{
    int x,y,z;
    printf("ingrese numero a sumar: ");
    scanf("%d",&x);
    printf("ingrese numero a sumar: ");
    scanf ("%d",&y);

    z=a+b;
    printf("La suma es %d",z);
    getch();
    return 0;
}
```

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

int suma(int a, int b);

int main()
{
    int x,y,z;
    printf("ingrese numero a sumar: ");
    scanf("%d",&x);
    printf("ingrese numero a sumar: ");
    scanf ("%d",&y);

    /*Llamada a la función*/
    z=suma(x,y);
    printf("La suma es %d",z);
    getch();
    return 0;
}

int suma(int a, int b)

    /*Desarrollo de la función*/
{
    int total;
    total=a+b;
    return total;
}
```

Declaración de la Función
En esta parte se define que tipo de valor va a egresar la función y los parámetros (constituyen la vía a través de la cual la rutina interactúa con el exterior, intercambiando datos) que necesita para realizar dicha función.

Variables Locales

Asignación de valores a las variables locales

Sentencia de Llamada o de invocación
Invocación de la función antes declarada, con los parámetros que solicita, la función suma requiere dos elementos para que funcione.

Definición de la función (recuerde que en esta parte no se le pone “;” al final, el parámetro es el nombre de la variable que utiliza la función en forma interna para hacer uso de cada valor que le está pasando el que la llamó.

Desarrollo de la Función

La variable total es una variable local, por lo cual no es reconocida en el programa principal sólo en ésta función.

- **La sentencia de llamada está formada por el nombre de la función y sus argumentos (los valores que se le pasan) que deben ir recogidos en el mismo orden que la secuencia de parámetros del prototipo y entre paréntesis.**

- **Si la función no recibe parámetros (porque así esté definida), entonces se coloca después de su nombre los paréntesis de apertura y cerrado sin ninguna información entre ellos. Si no se colocan los paréntesis, se produce un error de compilación.**

- **El paso de parámetros en la llamada exige una asignación para cada parámetro. El valor del primer argumento introducido en la llamada a la función queda asignado en la variable del primer parámetro formal de la función; el segundo valor de argumento queda asignado en el segundo parámetro formal de la función; y así sucesivamente. Hay que asegurar que el tipo de dato de los parámetros formales es compatible en cada caso con el tipo de dato usado en lista de argumentos en la llamada de la función.**

- **El compilador de C no dará error si se fuerzan cambios de tipo de dato incompatibles, pero el resultado será inesperado .**

- **La lista de argumentos estará formada por nombres de variables que recogen los valores que se desean pasar, o por literales. No es necesario (ni es lo habitual) que los identificadores de los argumentos que se pasan a la función cuando es llamada coincidan con los identificadores de los parámetros formales.**

- **Las llamadas a las funciones, dentro de cualquier función, pueden realizarse en el orden que sea necesario, y tantas veces como se quiera, independientemente del orden en que hayan sido declaradas o definidas. Incluso se da el caso, bastante frecuente como veremos más adelante, que una función pueda llamarse a sí misma.**

```
#include <stdio.h>
/*Declaración de la función usando parámetros por  valor*/

int suma(int a, int b);

int main()
{
      int x,y,z;
      printf("ingrese numero a sumar: ");
      scanf("%d",&x);
      printf("ingrese numero a sumar: ");
      scanf("%d",&y);

/*Llamada a la función*/
      z=suma(x,y);
      printf("La suma es %d",z);
      return 0;
}
int suma(int  a, int b)

/*Desarrollo de la función*/
{
      int total;
      total=a+b;
      return total;
}
```

```
#include <stdio.h>
/*Declaración de la función usando parámetros por referencia*/

int suma(int *a, int *b);
int main()
{
    int x,y,z;
    printf("ingrese numero a sumar: ");
    scanf("%d",&x);
    printf("ingrese numero a sumar: ");
    scanf("%d",&x);

/*Llamada a la función*/
    z=suma(&x,&y);
    printf("La suma es %d",z);
    return 0;
}
int suma(int *a, int *b)

/*Desarrollo de la función*/
{
    int total;
    total=*a+*b;
    return total;
}
```

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

int suma(int a, int b)
        /*Desarrollo de la función*/
{
        int total;
        total=a+b;
        return total;
}

int main()
{
        int suma(int a, int b);
        int x,y,z;
        printf("ingrese numero a sumar: ");
        scanf("%d",&x);
        printf("ingrese numero a sumar: ");
        scanf("%d",&y);

        /*Llamada a la función*/
        z=suma(x,y);
        printf("La suma es %d",z);
        return 0;
}
```

PROCEDIMIENTO usando VOID

```
#include <stdio.h>
void imprimeValor();
int main()
{
        int contador = 0;
        contador++;
        printf("El valor de contador es: %d\n", contador);
        imprimeValor();
        printf("Ahora el valor de contador es: %d\n", contador);
        return 0;
}
void imprimeValor()
{
        int contador = 5;
        printf("El valor de contador es: %d\n", contador);
}
```

```
#include <stdio.h>
#define max 9
int sumoarreglo(int a[],int n);

/* Funcion sumoarreglo, devuelve la suma de un arreglo de enteros */

int sumoarreglo(int a[],int n)
{
        int suma=0;
        for (int i=0;i<n;i++)
                suma += a[i];
        return suma;
}

int main()
{
   int b[max]= {0, 4, 78, 5, 32, 9, 77, 1, 23};
   int z;
   z=0;
   z= sumoarreglo(b,max);
   printf ("La suma es: %d",z);

  return 0;
}
```

No se ha especificado el tamaño de a.El ordenador contará los elementos que hemos puesto entre llaves y reservará espacio para ellos. De esta forma siempre habrá el espacio necesario,

Agregar la siguiente función al programa anterior.

```
/* Funcion promedio
float promedio(int a[],int n)
{

        int suma=0;
        float prom;
        prom=0;
        for (int i=0;i<n;i++)
                suma += a[i];
        prom=suma/n;
        return prom;
}
```