UNIVERSIDAD TECNOLÓGICA NACIONAL

INSTITUTO NACIONAL SUPERIOR
DEL
PROFESORADO TÉCNICO

TECNICATURA EN INFORMÁTICA APLICADA

PROGRAMACIÓN I

Módulo 3

ING. PATRICIA CALVO y LIC. MÓNICA HENCEK

Módulo 3

Funciones

Las funciones son bloques de código com un nombre asociado que realizan una tarea en particular. En C todo se construye con funciones, (main es la funcion principal del programa).

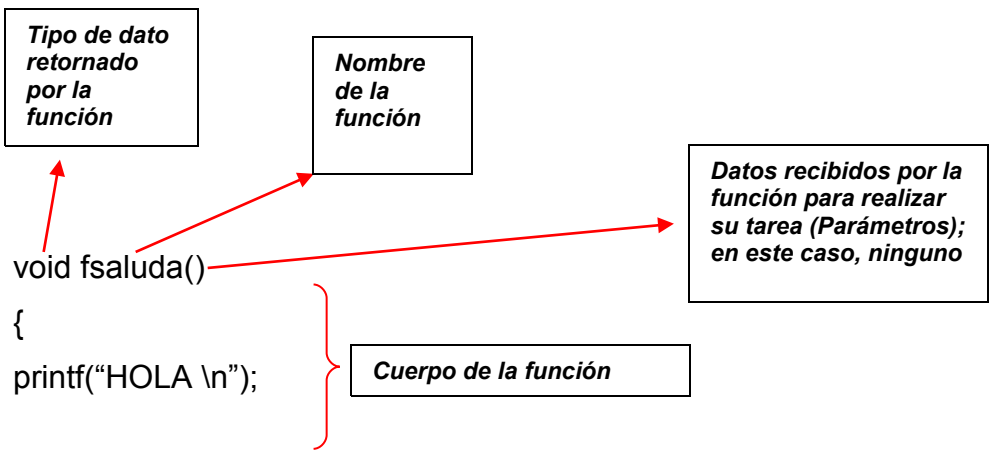
Una función se escribe una vez y luego se invoca, es decir, se ordena que se ejecute, todas las veces que sea necesario.

Algunas funciones requieren que se les pase uno o más datos para poder llevar a cabo su tarea; otras, no. Algunas retornan (es decir, devuelven) un dato, y otras no. En general, trataremos de que una función cumpla un único objetivo. Si al enunciar lo que debe hacer una función nos encontramos con que enumeramos tareas simples que son desarrolladas completamente por ella, lo más probable es que la función este mal planteada.

Por ejemplo, si una función tuviera que ingresar una secuencia de N números, calcular el mayor y emitirlo, no estaría bien diseñada, porque es poco probable que muchas veces se necesite realizar esa tarea compleja. Sin embargo, es cierto que muchas veces, se requiere ingresar una serie de números y almacenarlos, y que muchas veces se necesita obtener el valor mayor de un grupo de números ingresados previamente.

Entonces, lo correcto es escribir una función que permita ingresar y almacenar una secuencia de N números, escribir otra función que reciba una secuencia de N números y obtenga el mayor, y escribir una tercera función, que llame a las otras dos en el orden correcto.

Ejemplo: definición de función que emite un saludo por pantalla.

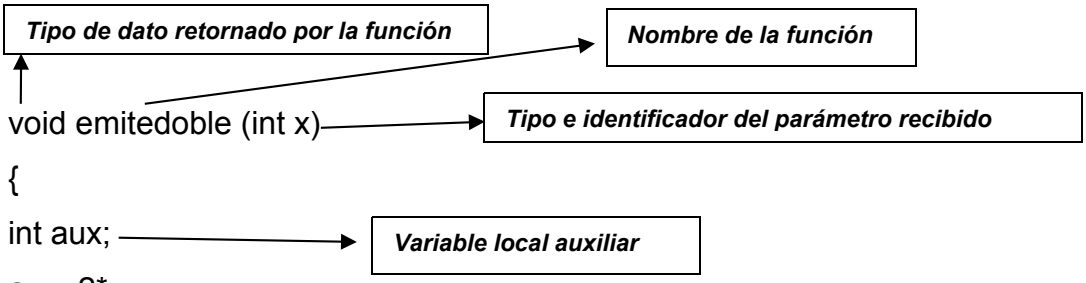


}

Ejemplo de invocacion a fsaluda:

```
int main()
{
fsaluda();
fsaluda();
printf("invocamos una vez mas\n");
fsaluda();
system("pause");
return 0;
}
```

Ahora veremos una función que recibe un dato entero y emite el doble del mismo por pantalla:



El parámetro x es una variable entera local, que se inicializa con el valor del argumento de llamada, aux es otra variable local pero no inicializada. Las variables locales solo existen mientras se esta ejecutando la función. Se generan en la pila o stack.

Ejemplo de uso:

```
int main()
{
int a,=5, b=8;                    //linea 1
```

```
emitedoble(5);                          //linea 2
emitedoble(8);                          //linea 3
emitedoble(10);                         //linea 4
printf("invocamos una vez mas");  //linea 5
emitedoble(a+b+1);                      //linea 6
system ("pause");
return 0;
}
```

Cuando se ejecuta la línea 2 y se invoca a emitedoble, el control del programa "sale" de la función main y pasa a la función emitedoble. Lo mismo ocurre en las líneas 3, 4 y 6. En cada caso, al terminar la ejecución de la función invocada, el control regresa a sentencia siguiente de la invocación. Esto es posible porque se almacena, también en la pila o stack, la dirección correspondiente a la siguiente instrucción, para que pueda continuarse la ejecución de main.

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (antes de la ejecución de emitedoble(a))
main	a= 5 b=8	----------

Cuando se ejecuta la linea 2, el control pasa a emitedoble, y la situación es:

control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (al comenzar la ejecución de emitedoble(a))
emitedoble	a= 5 b=8	x=5 aux

Al finalizar la ejecución de emitedoble(a), el estado de las variables es:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (un instante antes de finalizar la ejecución de emitedoble(a))
emitedoble	a= 5 b=8	x=5 aux =10

Cuando el control vuelve a main, antes de la llamada emitedoble(b), la situación es:

Módulo 3

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (antes de la ejecución de emitedoble(b))
main	a= 5 b=8	----------

Cuando se invoca a emitedoble(b), sucede esto:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (al comenzar la ejecución de emitedoble(b))
emitedoble	a= 5 b=8	x=8 aux

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (un instante antes de finalizar la ejecución de emitedoble(b))
emitedoble	a= 5 b=8	x=8 aux =16

Cuando el control vuelve a main, antes de la llamada emitedoble(10), la situación es:

control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (antes de la ejecución de emitedoble(10))
main	a= 5 b=8	----------

Luego se invoca a emitedoble(10), y es:

Cuando se invoca a emitedoble(b), sucede esto:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (al comenzar la ejecución de emitedoble(10))
emitedoble	a= 5 b=8	x=10 aux

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (un instante antes de finalizar la ejecución de emitedoble(10))
emitedoble	a= 5 b=8	x=10 aux =20

Módulo 3

Cuando el control vuelve a main, antes de la llamada emitedoble(10), la situación es:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (antes de la ejecución de emitedoble(a+b+1))
main	a= 5 b=8	----------

Finalmente se invoca a emitedoble(a+b+1), resultando:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (al comenzar la ejecución de emitedoble(a+b+1))
emitedoble	a= 5 b=8	x=14 aux

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (un instante antes de finalizar la ejecución de emitedoble(a+b+1))
emitedoble	a= 5 b=8	x=14 aux =28

Cuando el control vuelve a main, la situación es:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila
main	a= 5 b=8	----------

Como se observa, en ningún caso se altera el contenido de las variables definidas en main, llamadas variables estáticas.
Este modo de pasar los argumentos a una función se denomina pasaje por valor. La variable local parámetro, que corresponde al argumento de la función, se inicializa con una copia del valor del argumento correspondiente. Otro ejemplo:

Esta es una función que recibe dos enteros llamados m y n y emite por pantalla m%n si m>=n, o n%m en caso contrario (recordar que el operador % devuelve el resto entero de división entera).

Módulo 3

```
void resto (int m, int n)
{
//se ha elegido comparar a m con n para intercambiarlos en caso de que m<n
//aux es la variable local auxiliar para el intercambio
int aux;
if(m<n)
  {
    aux=m;
    m=n;
    n=aux;
  }
printf ("El resto de la division entre %d y %d es %d\n", m, n, m%n);
}
```

Ejemplo de invocaciones a resto:

```
int main()
{
 int a, b, c;
 a=100, b=27, c=39;
 resto(a,b);   //m corresponde a a y n a b
 resto (b,a);  //m corresponde a b y n a a
resto(23,c);   //m corresponde a 23 y n a c
resto (200, 49); //m corresponde a 200 y n a 49
resto(209+b-c, 25+a); //m corresponde a 209+b-c, y n a 25+a
system ("pause");
return 0;
}
```

Observar que la función resto puede recibir como argumento tanto variables, (de las cuáles hará copia de sus contenidos en sus variables locales) como valores *inmediatos,* (como por ejemplo 23, el cual se almacenará en el parámetro correspondiente). Cómo lograr que las funciones alteren el contenido de las variables correspondientes a los argumentos de llamada?.

Módulo 3

Supongamos que queremos una función que altere el contenido de una variable entera, duplicándolo. Para resolver esta cuestión, se pasará por valor la copia de la dirección de la variable. Entonces, la función en cuestión (llameémosla duplicar), recibirá un puntero a la variable, conteniendo la dirección de la variable. Es decir, que main podría ser, por ejemplo:

```
int main()
{
int s=4, h=18;
printf(" s vale %d\n", s)
//Le pasaremos a duplicar la dirección de s para que duplicar modifique su contenido
duplicar(&s);
printf("ahora s vale %d\n", s)
printf(" h vale %d\n", h)
//Ahora le pasaremos a duplicar la direccion de h para que duplicar modifique su contenido
duplicar(&h);
printf("ahora h vale %d\n", h)
system("pause");
return 0;
}
```

El código de duplicar puede ser, por ejemplo, (observar que se declara que duplicar recibe un puntero):

```
void duplicar (int *p)
{
  *p=(*p) *2;
}
p es &s cuando se invoca duplicar (&s)
p es &h cuando se invoca duplicar (&h)
```

Considerando, para ejemplificar, que &s sea 1000 y que &h sea 1004, es

Módulo 3

control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (antes de la ejecución de duplicar(s))
main	s= 4 h=18	----------

Cuando el control pasa a duplicar(s), la situación es:

control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (al comenzar la ejecución de duplicar(s))
duplicar	s= 4 h=18	p=1000 (p contiene l adireccion de s, y además *p vale **4**)

Al finalizar la ejecución de duplicar(s), el estado de las variables es:

control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (un instante antes de finalizar la ejecución de emitedoble(a))
duplicar	s= **8** h=18	p=1000 (ahora *p vale **8**)

Cuando el control vuelve a main, antes de la llamadaduplicar(h), la situación es:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (antes de la ejecución de duplicar(h))
main	s= 4 h=18	----------

Cuando el control pasa a duplicar(s), la situación es:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (al comenzar la ejecución de duplicar(s))
duplicar	s= 4 h=18	p=1004 (p contiene la dirección de h y además *p vale 18)

Al finalizar la ejecución de duplicar(s), el estado de las variables es:

Control	Estado de las variables de la Memoria estática	Estado de las variables de la Pila (un instante antes de finalizar la ejecución de emitedoble(a))
duplicar	s= 8 h=**36**	p=1000 (ahora *p vale **36**)

Es decir, como en el caso anterior, los parámetros de la función son variables locales, que solo existen mientras se ejecuta la función, pero como se

trata de un puntero, mediante ese puntero se puede alterar el valor de la variable de llamada.

Observar que en este caso, carece de sentido invocar a la función pasándole un valor *inmediato* (como duplicar(100), por ejemplo), ya que la función exige por su diseño que se le pase la dirección de una variable para modificar su contenido.

Resumiendo

Memoria estática	**Pila o Stack**
Aquí se ubican las variables definidas en main, que existen hasta la finalización de la ejecución de la misma.	Aquí se generan las variables de cada función que se invoca. Sólo existen mientras se ejecute esa función. Los parámetros son variables locales inicializadas - con el contenido de una variable o bien -con la dirección de una variable

Otro ejemplo: una función que permite ingresar un valor por teclado, se espera que la función permita leer un float y pueda ser invocada así:

```
int main()
{
  float f;
  leer(&f);
  printf ("La variable f contiene %f\n");
  system("pause");
  return 0;
}
```

La función puede ser:

```
void  leer(float *q)
{
printf("Ingrese un valor real\n");
```

```
scanf("%f", q);  //observar que se puede usar q para ingresar el real, ya que es
la dirección en donde debe almacenarse el valor entrante
system ("pause");
return 0;
}
```

Funciones que retornan un valor:

Ahora se diseñará una función que reciba tres valores enteros y retorne el promedio de los mismos. El valor devuelto por la función es float. La función (llamémosla prom) podrá invocarse de estos modos:

```
int main ()
{
int a=4,b=10,c=8;
float r;
r = prom(a,b,c);
printf("El promedio entre %d, %d y %d es %f \n",a,b,c,r);
r= prom(100, 34, a);
printf("El promedio entre %d, %d y %d es %f \n",100,34,a,r);
printf (("El promedio entre %d, %d y %d es %f \n",70,30,b, prom(70,30,b));
if(prom(3,a,4)>5) printf ("El promedio obtenido es mayor que 5\n");
    else printf ("El promedio obtenido es menor que 5\n");
r=prom (34,56,a) + prom (a,b,c) - 100;
printf("El resultado de efectuar prom (34,56,a) + prom (a,b,c) - 100 es %f\n",r)
system ("pause");
return 0;
}
```

Observar que la invocación a prom se utiliza como si fuera una expresión float, y debe, o bien ser almacenada en una variable, o ser usada en una expresión aritmética, o para construir una expresión lógica, o directamente en un printf. Nada de esto era posible con las funciones que devolvían void. La función prom puede diselarse así:

Tipo de dato retornado por prom

Módulo 3

```
float prom(int x, int y , int z)
{
float aux=(x+y+z)/3;
return aux;
}
```

Toda función que retorne un tipo de dato diferente de void debe tener al menos una línea return.

Otro ejemplo: otra forma de ingresar un valor desde teclado, si diseñamos una función del siguiente modo:

```
int lee()
{
int aux;
printf("Ingrese un valor entero");
scanf("%d", &aux);
return aux;
}
```

La variable aux es local; sólo existe mientras se ejecuta lee(). La dirección de aux corresponde a alguna posición dentro de la zona de pila. El valor almacenado en aux se retorna. La invocación puede ser :

```
int main()
{
int a,b;
a=lee();
printf("a vale %d\n", a);
b=lee();
printf("b vale %d\n", b);
a=lee();
printf("Ahora a vale %d\n", a);
return 0;
}
```

Módulo 3

3.1 Ejercicios:

3.1.1) Escribe la función booleana PRIMO, que devuelve true si un número dado es primo.

3.1.2) Escribe la función booleana MULTIPLO, que recibe dos valores enteros y devuelve true si el primero es múltiplo del segundo.

3.1.3) Escribe la función FIBO, que devuelve el n-ésimo término de la sucesión de Fibonacci: Sucesión de Fibonacci: 1,1,2,3,5,8,13,...

3.1.4) Escribe una función o procedimiento para calcular la potencia de dos enteros.

3.2 Funciones recursivas

1) Escribe funciones recursivas que retornen:

a) El producto de dos números enteros recibidos ,en función de la suma.

b) La división entera de dos números enteros recibidos por parámetro en función de la resta.

c) El máximo común divisor entre dos números enteros.

d) El factorial de un número entero

g) Un valor booleano indicando si un número dado por parámetro es capicúa.

e) El n-ésimo término de la sucesión de Fibonacci, recibido el n.

2)Escribir procedimientos que realicen el mismo trabajo que cada una de las funciones anteriores.

3) Escribe un programa que lea un número entero correspondiente a la cantidad de discos a usar y emita los movimientos correspondientes al juego de las Torres de Hanoi.

4) Escribe una función recursiva que reciba un vector de enteros, y el N (numero de componentes del vector) y emita la suma de sus componentes.

5) Escribe el módulo anterior usando un procedimiento.

6) Escribe un procedimiento recursivo que lea un par de enteros (analiza las restricciones) y emita el resultado de elevar el primer número a la potencia indicada por el segundo.

7) Escribe una función o procedimiento recursivo que reciba un vector de N componentes, y el N, y retorne un valor booleano indicando

Módulo 3

a) Si hay algún cero en el vector

b) Si todas las componentes son ceros

c) Si todas las componentes son múltiplos del último elemento

d) Si alguna es múltiplo del último elemento

8) Emite un vector ‘al revés’ recursivamente

9) Escribe un módulo recursivo que retorne la productoria de los elementos de un vector de enteros

10) Escribe un módulo recursivo que reciba un string y retorne otro que sea la inversión del primero.

11) Escribe una función booleana recursiva que reciba dos strings y retorne valor indicando si son iguales

12) Idem 11 pero indicando si el primer string esta contenido en el segundo.

13) Escribe un módulo recursivo para la Búsqueda Binaria en un array.