PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

MSc. Alexander Prieto León

Conferencia II

Unidad I Programación estructurada para aplicaciones de automatización.

1.2 Estructuras de control de flujo y funciones.

Objetivos:

- Usar Estructuras de Control Alternativas para resolver problemas en C.
- Usar Estructuras de Control Repetitivas para resolver problemas en C.
- Emplear funciones de librerías dentro de programas en C.
- Implementar funciones propias que contribuyan a resolver problemas en C.

- **Bibliografía:**

  - Deitel and Deitel. Como programar en C/C++. Segunda edición o superior. (preferiblemente la 4ta edición)

  - De la Fuente y otros. Aprenda lenguaje ANSI C como si estuviera en primero. Universidad de Navarra.

- **En la clase anterior:**
- **¿Qué elementos de programación en C tratamos?**
- **¿Cuáles son los tipos de datos simples?**
- **¿Qué operadores tratamos?**
- **¿Cómo se crea una constante?**
- **¿Cuándo es necesario realizar type cast explícito?**

- **¿Si queremos programar la siguiente función matemática qué problema encontramos?**

$$F(x) = \frac{3(x^2+1)}{x-6}$$

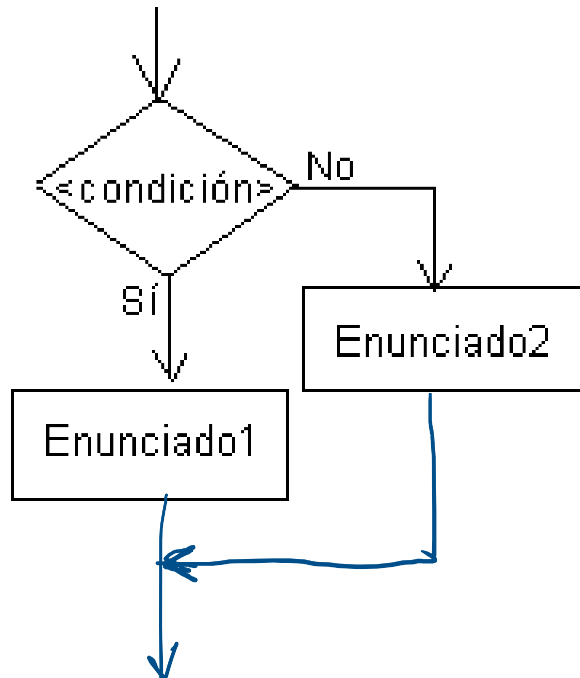
- **¿Hasta ahora en que orden son ejecutadas las instrucciones de todos los programas que hemos visto?**
  - Ejemplo:

```
main()
{
   const float pi=3.14; //Constante pi
   int r,h;
   float Ab,Vb;
   printf("introduzca el radio y la altura:");
   scanf("%d %d",&r, &h);
   Ab = pi*r*r;
   Vb= Ab*h;
   printf("El volumen es: %f",Vb );
}
```

Estructura de Control Condicional

- Estructura condicional **if-else**

Diagrama en Bloques:



Lenguaje C:

```
if (condición)
{
   Enunciado1;
}
else
{
   Enunciado2;
}
```

Estructura de Control Condicional

- Estructura condicional **if-else**

Lenguaje C:

```
if (condición)
{
   Enunciado1;
}
else
{
   Enunciado2;
}
```

- Observaciones:
  - **if** y **else** son palabras claves del lenguaje.
  - Condición: se evalúa como una expresión lógica. O sea, el resultado de evaluar dará 0 ó 1
  - Las llaves {} seguidas después de la condición definen el bloque de sentencias que se ejecutarán si "condición" es 1
  - Las llaves {} seguidas después la palabra **else** definen el bloque de sentencias que se ejecutarán si "condición" es 0
  - Enunciado1 y Enunciado2 pueden contener más de una instrucción
  - La estructura (**if-else**) puede anidarse una dentro de otra.

Estructura de Control Condicional

- Estructura condicional **if-else**
- **Ejemplo 1:**

- Crear un programa que permita evaluar la función matemática:

$$F(x) = \frac{3(x^2+1)}{x-6}$$

Estructura de Control Condicional

- Estructura condicional **if-else**
- **Ejemplo 1:**

```
#include <stdio.h>
main()
{
    float x;
    scanf("%f",&x);
    if (x != 6)
    {
        printf("F(%f)=%f",x, 3*(x*x+1)/(x-6) );
    }
    else
    {
        printf("no es posible evaluar F(x)");
    }
}
```

Estructura de Control Condicional

- Estructura condicional **if-else**
- **Ejemplo 2:**

- Crear un programa que permita evaluar la función matemática:

$$F(x) = \frac{3(x^2+1)}{(x-6)(x+2)}$$

Estructura de Control Condicional

- Estructura condicional **if-else**
- **Ejemplo 2 (variante 1):**

```
#include <stdio.h>
main()
{
    float x;
    scanf("%f",&x);
    if (x == 6 )
        printf("no es posible evaluar F(x)");
    else
        if(x == -2)
            printf("no es posible evaluar F(x)");
        else
            printf("F(%f)=%f",x, 3*(x*x+1)/((x-6)*(x-2)) );
}
```

```
//Otra forma de organizar parecida al
//elsif de otros lenguajes
#include <stdio.h>
main()
{
    float x;
    scanf("%f",&x);
    if (x == 6 )
        printf("no es posible evaluar F(x)");
    else if(x == -2)
        printf("no es posible evaluar F(x)");
    else
        printf("F(%f)=%f",x, 3*(x*x+1)/(…
}
```

Estructura de Control Condicional

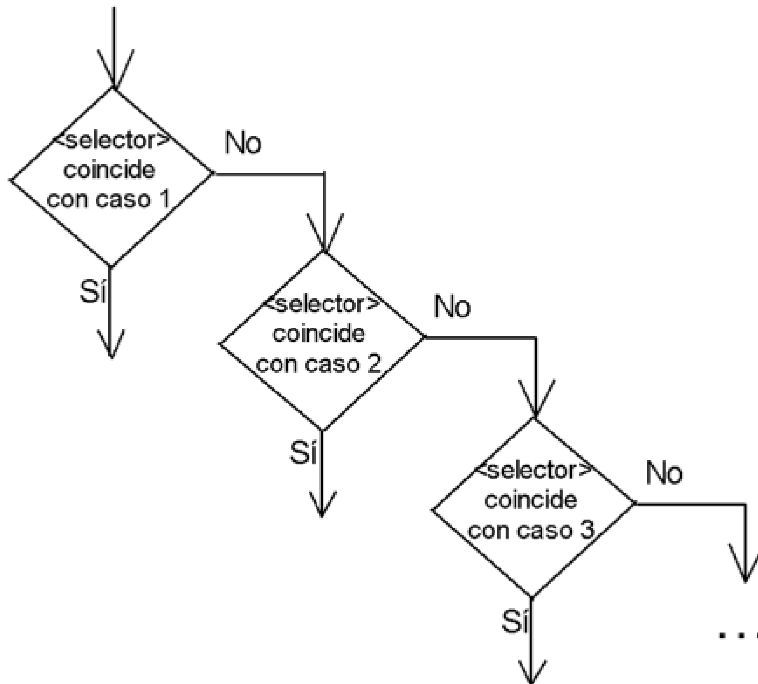
- Estructura condicional **if-else**
- **Ejemplo 2 (variante 2):**

```
#include <stdio.h>
main()
{
    float x;
    scanf("%f",&x);
    if (x != 6 && x != -2   )
    {
        printf("F(%f)=%f",x, 3*(x*x+1)/(x-6)*(x-2) );
    }
    else
    {
        printf("no es posible evaluar F(x)");
    }
}
```

Estructura de Múltiple Selección

- Estructura de múltiple selección **switch**

Diagrama en Bloques:



Lenguaje C:

```
switch (selector)
{
case  valor1:
        enunciado 1;
        break;
case  valor2:
        enunciado 2;
        break;
:
:
case  valorN-1:
        enunciado n-1;
        break;
default:
        enunciado N;
        break; //opcional, igual terminará
}
```

Estructura de Múltiple Selección

- Estructura de múltiple selección **switch**
  - **switch, case**, **break** son palabras claves del lenguaje C
  - selector es una variable que puede ser de tipo **char, short, int, long.**
  - valor1, valor2..., valorN-1 son los posibles valores que puede tomar la variable identificada como selector.
  - enunciado 1 se ejecutará en caso de que la variable selector tome el valor valor1
  - enunciado 2 se ejecutará en caso de que la variable selector tome el valor valor 2

```
switch (selector)
{
case  valor1:
        enunciado 1;
        break;
case  valor2:
        enunciado 2;
        break;
:
:
case  valorN-1:
        enunciado n-1;
        break;
default:
        enunciado N;
        break;
}
```

Estructura de Múltiple Selección

- Estructura de múltiple selección **switch**
  - enunciado n-1 se ejecutará en caso de que la variable selector tome el valor valorN-1
  - enunciado N se ejecutará en caso de que la variable selector tome un valor diferente a los casos anteriores
  - La instrucción **break** sirve para indicar que una vez se ejecute uno de los enunciados se salga de la estructura **switch.** Si no se coloca la misma entonces se ejecutaran lo enunciados que están después de aquel para el que se cumplió la condición:
    - selector = valorX

```
switch (selector)
{
case  valor1:
      enunciado 1;
      break;
case  valor2:
      enunciado 2;
      break;
:
:
case  valorN-1:
      enunciado n-1;
      break;
default:
      enunciado N;
      break;
}
```

Estructura de Múltiple Selección

- Estructura de múltiple selección **switch**
- **Ejemplo**: Se quiere hacer un programa que permita evaluar 3 funciones matemáticas diferentes:

$$f(x) = x^2$$

$$g(x) = x^3$$

$$h(x) = x^4$$

- Requerimientos: El programa deberá presentar en pantalla la información de las funciones y el usuario deberá seleccionar la ecuación introduciendo por teclado f, g, o h. En caso de no ser ninguna de estas letras, debe mostrarse un mensaje de alerta y salir del programa. Si es una de las letras válidas entonces se pide al usuario el valor de x que se quiere evaluar, se imprime en pantalla el resultado y se sale del programa.

Estructura de Múltiple Selección

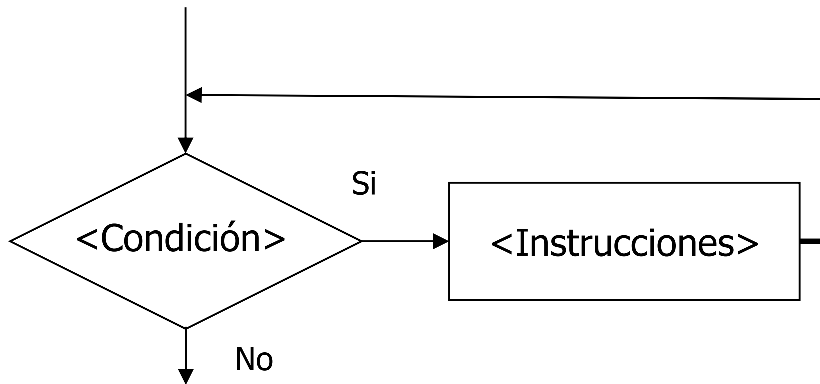
- Estructura de múltiple selección **switch**
- **Ejemplo**:

```
#include <stdio.h>
main()
{
char c;
float x;
printf("Seleccione la función usando f,g,h\n");
printf("f(x) = x^2 \ng(x) = x^3\nh(x) = x^4\n");
scanf("%c", &c);
switch(c)
{
case 'f': printf("entre x:");
          scanf("%f",&x);
          printf("\nx^2=%f",x*x);
          break;
case 'g': printf("entre x:");
          scanf("%f",&x);
          printf("\nx^3=%f",x*x*x);
          break;
case 'h': printf("entre x:");
          scanf("%f",&x);
          printf("\nx^4=%f",x*x*x*x);
          break;
case '\n':  //Para '\n', '\t' y ' ' se ejecutará
case '\t':  // la misma instrucción break
case ' ':break;
default:
          printf("Opción no válida");
}
getchar();
}
```

Estructura while

- Representa un ciclo con precondición.

- **Diagrama en Bloques:**



Lenguaje C:

```
while (condición)
{
    Instrucciones
}
```

Estructura while

- Representa un ciclo con precondición.

  - **while** es palabra clave del lenguaje.
  - Condición: se evalúa como una expresión lógica. O sea el resultado de evaluarla dará "0" ó "1"
  - Las llaves {} seguidas después de la condición definen el bloque de instrucciones que se ejecutarán mientras condición de como resultado "1".
  - En este caso puede pasar que el ciclo nunca se realice.

Lenguaje C:

```
while (condición)
{
    Instrucciones
}
```

Estructura while

Ejemplo:

- Realice un programa que permita obtener de teclado un número entero positivo n, e imprima en pantalla todos los números positivos menores que n.

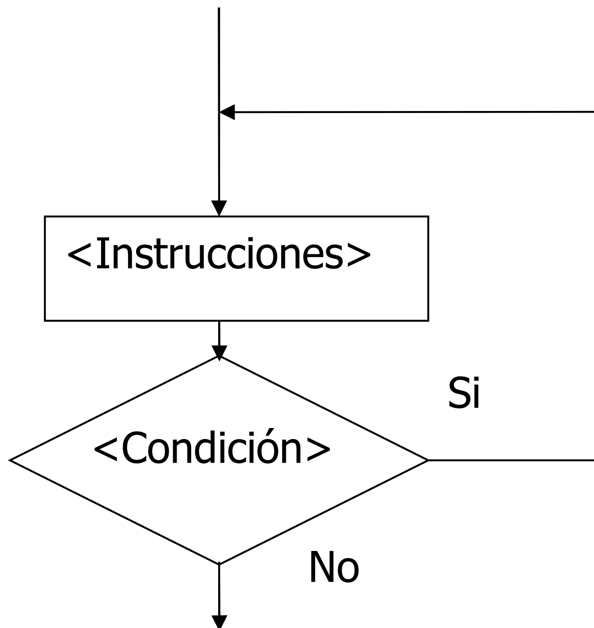
```
#include <stdio.h>
main()
{
  int n, i=0;
  scanf("%d", &n);
  while (i < n)
  {
     printf("%d\n", i);
     i++;
  }
}
```

```
#include <stdio.h>
main()
{
  int n, i=0;
  scanf("%d", &n);
  while (i < n)
     printf("%d\n", i++);
 }
```

Estructura do-while

- Representa un ciclo con poscondición.

- **Diagrama en Bloques:**



Lenguaje C:

```
do
{
    Instrucciones
}
while (condición)
```

Estructura do-while

- Representa un ciclo con poscondición.

  - **do, while** son palabras claves del lenguaje.
  - Condición: se evalúa como una expresión lógica. O sea el resultado de evaluarla dará "0" ó "1"
  - Las llaves {} seguidas después de la condición definen el bloque de instrucciones que se ejecutaran mientras condición de cómo resultado "1".
  - En este caso el ciclo se repetirá al menos una vez

Lenguaje C:

```
do
{
    Instrucciones
}
while (condición)
```

Estructura do-while

Ejemplo:

- Realice un programa que permita obtener de teclado un número entero positivo n, e imprima en pantalla todos los números pares desde 0 hasta n.

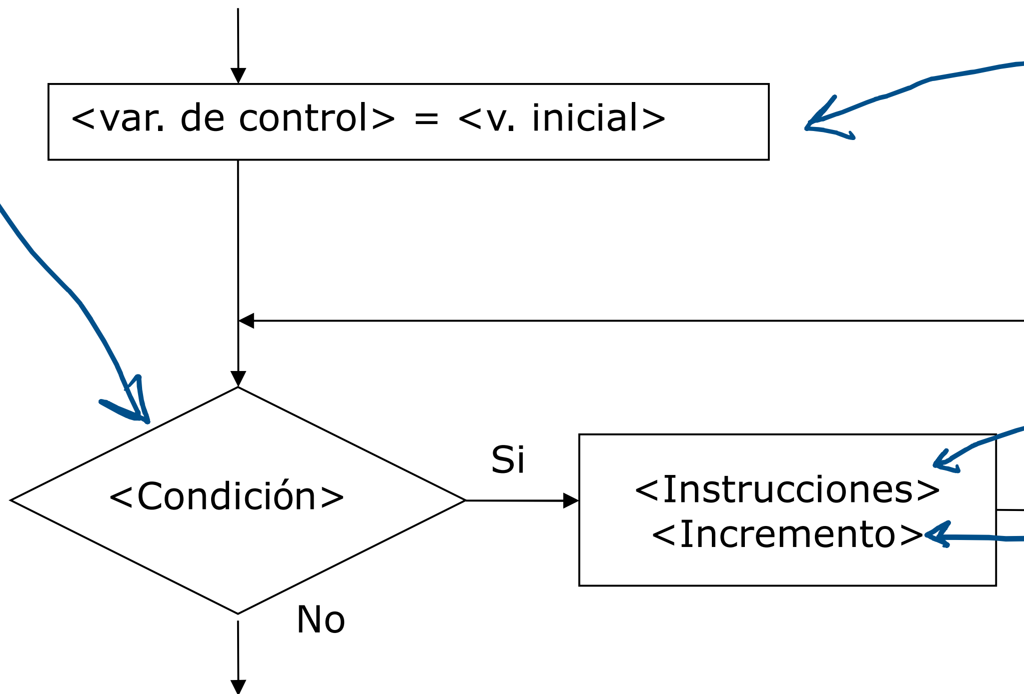
```
#include <stdio.h>
main()
{
int n, j=0;
scanf("%d", &n);
do
{
  printf("%d",j);
  j= j+ 2;
}
while (j<= n);
}
```

```
#include <stdio.h>
main()
{
int n, j=0;
scanf("%d", &n);
do
  printf("%d", j+=2);
while (j<= n);
}
```

Estructura for

- Representa un ciclo por variable de control.

- **Diagrama en Bloques:**



Lenguaje C:

```
for (exp1; exp2; exp3)
{
   Enunciado1;
   Enunciado2;
   EnunciadoN;
}
```

Estructura for

- Representa un ciclo por variable de control.
  - **for**: Es palabra clave de lenguaje C
  - expresion1: fase de inicialización;
  - expresion2: condición lógica, verificación de continuación de ciclo
  - expresion3: incremento de variable de control
  - Equivale a:

```
expresión1;
while (expresión2)
{
    Enunciado1;
    Enunciado2;
    EnunciadoN;

    expresion3;
}
```

Lenguaje C:

```
for (exp1; exp2; exp3)
{
    Enunciado1;
    Enunciado2;
    EnunciadoN;
}
```

Estructura for

- Representa un ciclo por variable de control.
  - expresion1 y expresion3 son expresiones que pueden ser a menudo contenedoras de varios enunciados que se separan por comas. Expresiones en listas separadas por comas.
  - Las 3 expresiones de la estructura **for** son opcionales. Si se omite expresión2, C supondrá que la condición es verdadera creando por lo tanto un ciclo infinito.
  - También se puede omitir la expresión1 si la variable se inicializa en alguna otra parte del programa antes de **for**. La expresion3 también se puede omitir si la variable se incrementa mediante enunciados en el cuerpo de la estructura, o si no se requiere de incrementos;
  - Los ";" son requeridos.

Lenguaje C:

```
for (exp1; exp2; exp3)
{
    Enunciado1;
    Enunciado2;
    EnunciadoN;
}
```

Estructura for

Ejemplo:

- Realice un programa que sume los n primeros números enteros positivos.

<table>
<tr>
<td>

```
#include <stdio.h>
main()
{
int numero,i,suma;
scanf("%d", & numero);
for (i=0,suma=0;i<= numero;i++)
        suma += i;

printf("la suma es: %d", suma);
}
```

</td>
<td>

```
#include <stdio.h>
main()
{
int numero,i=0,suma=0;
scanf("%d", & numero);
for ( ;i<= numero;i++, suma += i);

printf("la suma es: %d", suma);
}
```

</td>
</tr>
</table>

Estructura for

- Observaciones:
  - Tanto la inicialización, como la condición de continuación de ciclo, y el incremento pueden contener expresiones aritméticas.

    for(j=x; y < 4*x*y; j+=y/x)

    para x=2 y y = 10 equivale a:

    for(j=2; y < 80; j+=5)

  - El incremento puede ser negativo (realmente se decrementa) el ciclo va hacia atrás.
  - Si la condición de continuación de ciclo resulta falsa al inicio, la porción del cuerpo no se ejecutará.
  - Se puede utilizar el operador ", "para crear expresiones en listas

    Ej: **for** (;i<= numero;i++, suma += i); // variante 2

Enunciados continue y break

- **continue:** Ejecuta la iteración siguiente o evalúa la condición
- **break:** Sale de la estructura repetitiva o de múltiple selección

- **Ej:** Realice un programa que permita obtener de teclado un número entero positivo n, e imprima en pantalla todos los números pares desde 0 hasta n, excepto el 5.

```
#include <stdio.h>
main()
{
  int n, i;
  scanf("%d", &n);
  for (i=0;i<=n; i++)
  {
       if (i==5)
           continue;  //Salta iteración
       printf("%d\n",i);
  }
}
```

Funciones

- ¿Qué funciones de C hemos usado?

Funciones

- **¿Qué funciones de C hemos usado?**
- printf(), scanf(), getchar()
- Quien llama a una función no conoce cómo esta realiza su tarea, solo conoce que la **necesita para realizar determinada tarea y lo que devuelve esta (en caso de que devuelva algún resultado).**

- **Funciones**

- Programas grandes
  - Divide y vencerás
  - Módulos (en C-> funciones)
    - Funciones
      - Biblioteca estándar
      - Definidas por el usuario

- Las funciones se invocan mediante llamada a función
  - Nombre de la misma
  - Se les proporciona información a través de los argumentos, para que pueda realizar su tarea.

Funciones

Ventajas:

- El programa es más manipulable.
- Reutilización de software. (printf, scanf)
- Evitar repetición de código.
- El tiempo de diseño se acorta.
- Depuración de código

Funciones de biblioteca o librería

- Para calcular la raíz cuadrada la función es la siguiente:

double sqrt(**double**); //en biblioteca math.h

Qué indica que la función recibe como parámetro una variable de tipo double y regresa otra de tipo double. El resultado es la raíz cuadrada del valor recibido como parámetro.

Ej: x=sqrt(9) //x=3

- Otra función interesante es:

double pow **(double, double);** //en biblioteca math.h

Esta función devuelve como resultado el primer parámetro elevado al segundo parámetro.

Ej: x=pow(3,2) //x=9

Funciones de biblioteca o librería

- Además, tenemos las funciones trigonométricas, logarítmicas y exponenciales.

PROTOTIPO	FUNCIÓN MATEMÁTICA
double sin (double);	seno(x)
double cos (double);	coseno(x)
double tan (double);	tangente(x)
double log (double);	log(x)
double exp (double);	e^x

- En la bibliografía puede encontrar todas las funciones matemáticas.

Funciones de biblioteca o librería

- Ejemplo: Calcular las raíces de una ecuación de 2do grado. Los coeficientes se introducen por teclado.

```
#include <stdio.h>
#include <math.h>
int main()
{
    float a,b,c,d;
    printf("Entre (a,b y c) de ax^2+bx+c:");
    scanf("%f %f %f", &a,&b,&c);
    d = pow(b,2)-4*a*c;
    if (d<0)
        printf("no tiene solución\n");
    else
        printf("las soluciones son: %.2f y %.2f\n",(-b-sqrt(d))/(2*a),(-b+sqrt(d))/(2*a));
}
```

Llamada a la función pow

Las funciones se invocan por su nombre.

Los argumentos se pasan seguido del nombre y entre paréntesis, separados por coma en caso de ser más de uno.

El llamado a una función devuelve una variable que es del tipo indicado por su prototipo. Este valor puede ser usado directamente o asignado a una variable auxiliar.

Implementación de Funciones

- Además de las funciones de biblioteca, el programador puede definir sus propias funciones y definir su propia biblioteca que pudiera usar cada vez que se necesite.

- **Formato de definición de una función**

```
[Tipo de regreso] NOMBRE_FUNCION ([Lista de parámetros sep. por comas])
{
    Declaración de variables
    Enunciados
    return [variable/expresión]
}
```

Implementación de Funciones

Formato de definición de una función

```
[Tipo de retorno] NOMBRE_FUNCION ([Lista de parámetros sep. por comas])
{
    Declaración de variables
    Enunciados
    return [variable/expresión]
}
```

- El nombre de función es cualquier identificador válido (no espacios, ni operadores de C)
- El tipo de retorno debe ser del mismo tipo que [variable/expresión].
- **void** – Tipo especial que indica que la función no devolverá VALOR.
- La lista de parámetros consiste en una lista separada por comas que contiene las declaraciones de los parámetros recibidos por la función al ser llamada. Si una función no recibe ningún valor, la lista de parámetros es **void**.

Implementación de Funciones

Formato de definición de una función

```
[Tipo de retorno] NOMBRE_FUNCION ([Lista de parámetros sep. por comas])
{
    Declaración de variables
    Enunciados
    return [variable/expresión]
}
```

- El cuerpo de la función está definido por {}
- Bajo ninguna circunstancia puede ser definida una función dentro de la definición de otra.
- Existen tres formas de terminar la ejecución de una función.
  - }
  - return;
  - return variable/expresión;

Implementación de Funciones

- **Ejemplo 1:** función que suma 3 números enteros.

Código	Descripción
`#include <stdio.h>`	
`int suma3int(int,int,int);`	**Prototipo de la función**: ha de estar definido antes de la llamada a la función en main(). Se puede obviar si en su lugar se pone la implementación de la función.
`main()`	
`{`	
`int n1,n2,n3,S;`	
`printf("Entre los tres numeros a sumar:");`	
`scanf("%d %d %d", &n1,&n2,&n3);`	
`S = suma3int(n1,n2,n3);`	**Llamada a la función** guardando su valor de retorno en S
`printf("El valor de la suma es: %d", S);`	
`}`	
`int suma3int(int a,int b,int c)`	**Implementación de la función** suma3int()
`{`	
`return a+b+c;`	
`}`	

```c
#include <stdio.h>
int suma3int(int,int,int);

main()
{
    int n1,n2,n3,S;
    printf("Entre los tres numeros a sumar:");
    scanf("%d %d %d", &n1,&n2,&n3);
    S = suma3int(n1,n2,n3);
    printf("El valor de la suma es: %d", S);
}

int suma3int(int a,int b,int c)
{
    return a+b+c;
}
```

Implementación de Funciones

- **Ejemplo 1:** función que suma 3 números enteros.

``
#include <stdio.h>
int suma3int(int,int,int);
 	
main()
{
    int n1,n2,n3,S;
    printf("Entre los tres numeros a sumar:");
    scanf("%d %d %d", &n1,&n2,&n3);
    S = suma3int(n1,n2,n3);
    printf("El valor de la suma es: %d", S);
}
 	
int suma3int(int a,int b,int c)
{
    return a+b+c;
}
``
	A **n1, n2, n3** al introducirse entre los paréntesis de sum3int(), en la llamada, se les dice argumentos de la función. A **int a,int b,int c** en la implementación de suma3int() se les llama parámetros de la función. **a, b y c** son variables locales en la función, se crean con la llamada y se destruyen al finalizar la llamada.
	
	En la llamada a la función ocurre: **a=n1**, **b=n2** y **c=n3**. Esto se llama paso de parámetros por valor.

Implementación de Funciones

- **Ejemplo 2:** Defina una función en C que permita evaluar la siguiente función matemática:

$$f(x) = \begin{cases} \sin x^5 \ \boldsymbol{si} \ x < -1 \\ e^{-x} si -1 \le x \le 1 \\ \sqrt[3]{x} \ si \ x > 1 \end{cases}$$

```
float f(float x)
{
   if (x < -1)
       return sin(pow(x,5));
   else if (x <= 1)
       return exp(-x);
   else
       return pow(x, 1.0/3.0);
}
```

Implementación de Funciones

- **Ejemplo 2:** Escriba un programa principal donde se pida al usuario un valor real a y se devuelva el resultado de evaluar la función f(a) anterior:

```
#include <stdio.h>
#include <math.h>
float f(float);

int main()
{
    float a,b;
    printf("Entre la abscisa:");
    scanf("%f", &a);
    b = f(a);
    printf("El valor f(%f)=%f",a,b);
}
```

Aquí va la implementación de la función.

Si la definición de función está antes de main, no es necesario incluir el prototipo. Si está después es obligatorio.

Cuando se invoca la función **float** f(**float**) se crea otra variable x que es local a la función f y se destruye al finalizar la llamada a la función.

La variable x es una variable local a la función f.

Esto se denomina paso de parámetros por valor y se copia el valor de una en la otra. Una función puede invocar a tantas funciones cómo necesite.

Implementación de Funciones

Alcance de las variables:

- **Variables locales:** Una variable es local a un bloque cuando se declara al inicio de este. El bloque puede ser de definición de función o de cualquier estructura condicional o repetitiva. Cuando se termina el bloque se destruye la variable. Todas las variables usadas hasta el momento han sido locales a main().

- **Variables globales:** Se definen fuera de cualquier función y se pueden invocar desde cualquier función. Las variables globales se mantienen en memoria todo el tiempo de ejecución del programa. Al igual que las variables declaradas al inicio de *main*(). Las segundas solo son visibles a *main*(), las primeras son visibles a todas las funciones que el usuario defina.

Implementación de Funciones

Alcance de las variables: ejemplo

```
void getinterval(){
 printf("Entre un intervalo de enteros:");
 scanf("%d %d", &a, &b);
 if (a > b){
     int t = b;
     b = a;
     a = t;
 }
}

int suma(){
    int i,s=0;
    for(i=a;i<=b;i++)
       s += i;
    return s;
}
```

t es variable **local** al bloque **if**, se destruye cuando se ejecuta la llave de cierre indicada en amarillo }

```
#include <stdio.h>
#include <math.h>
int a,b;

int main()
{
  getinterval();
  printf("La suma de los enteros del intervalo
         [%d,%d] es: %d",a,b,suma());
}
```

a, b variables **globales** ya que están declaradas fuera de cualquier función

a, b son visibles en la definición de las funciones *getinterval*, *main* y *suma* por ser declaradas **globales**.

- Antes de main() deberían estar definidos los prototipos de las dos funciones

Implementación de Funciones

- Alcance de las variables: ejemplo

- Observaciones:
  - a y b son globales, por tanto están disponibles a todas las estructuras y funciones del programa.
  - t es local al bloque de instrucciones de la condición verdadera del if.
  - i,s son locales a la función suma.
  - **void getinterval()** no retorna valor
  - **int suma()** no recibe parámetros pero devuelve un entero que es la suma de los enteros del intervalo.

Implementación de Funciones

Clases de almacenamiento

- **auto**: son aquellas variables que se crean al comienzo de un bloque. Por defecto las variables son auto por lo que nunca se usa.
- **register**: registros de procesador. Se gana en eficiencia. Los compiladores modernos tratan de hacerlo automáticamente
- **extern**: son variables globales que se declaran fuera de cualquier función. La palabra clave es útil cuando se quiere usar la variable en una función que se definió antes de la declaración de variable. Por defecto las variables fuera de main() son extern.
- **static**: Son variables que nunca se destruyen (como las globales), pero solo son visibles en el bloque donde fueron declaradas.

Implementación de Funciones

- Clases de almacenamiento
- Ejemplo 1 (static): ¿Qué hace f()?

```
int f(int);

int main(){
        int i;
        for(i=0;i<10;i++)
          printf("%d \n",f(1));
        getchar();
        return 0;
}
int f(int a){
        static int d=0; // no se destruye al termino de  f
        d++;
        return d;
}
```

Implementación de Funciones

- Clases de almacenamiento
- Ejemplo 2 (extern): ¿Qué hace f()?

```
#include <stdio.h>
int c=4;
int f(int);
int main(){
        int i;
        for(i=0;i<10;i++)
          printf("%d \n",f(1));
        getchar();
        return 0;
}
int f(int a){
  int c=0;
  return ::c; // muestra la variable global;
}
```

CONCLUSIONES