

Control de flujo

Antonio Espín Herranz

Control de flujo del programa

- Sentencias de Control.
 - Condicionales: if, if else, switch.
 - Repetición: while, for.
- Anidación de bucles.
- Funciones y procedimientos.
- Paso de parámetros.
- Recursividad.
- Diseño de programas.

Condicionales if, if else, switch

```
if (condición){  
    Sentencia true;  
}
```

```
if (condición){  
    Sentencias true;  
} else {  
    Sentencias false;  
}
```

```
switch(expresion){  
    case op1: ...  
        break;  
    case op2:  
        break;  
    default:  
        break;  
}
```

Opi: números, 'a'.

switch

- **Ojo en switch:**

- Si hacemos alguna declaración de variable dentro del case, hay que añadir { }

```
switch(var){  
    case 1:  
        {  
            int i;  
            for (i = 0 ; ...)  
                ...  
            break;  
        }  
    case 2:  
        break  
}
```

Bucles while y for

- **Bucle for:**

```
for ([inicialización]; [condición] ; [incremento]){  
    Instrucciones;  
}  
for (i = 0 ; i < 100 ; i++)    for (;;);
```

- **Bucle while:**

```
while (condición){  
    instrucciones  
}
```

- **Bucle do while:**

```
do {  
    instrucciones;  
} while (condición);
```

Anidación de bucles

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

void main(void){
    int numero, n, resultado;
    for (numero=1 ; numero <=10;numero++){
        printf("\nTabla del %d:", numero);
        for (n=1 ; n <= 10 ; n++){
            resultado = n * numero;
            printf("\n\t%d x %d = %d",n,numero, resultado);
        }
        printf("\n Pulse una tecla para continuar");
        getch();
    }
}
```

break y continue

- **break:** Rompe el bucle.

```
for ( i = 0 ; i < 9 ; i++){  
    if (i == 5) break;  
    printf("\n %d", i);  
}
```

- **continue:** Fuerza otra iteración.

```
for (i = 0 ; i < 9 ; i++){  
    if (i == 5) continue;  
    printf("\n %d", i);  
}
```

Funciones y procedimientos

- Programación modular, nos permite dividir nuestro programa en partes independientes mas fáciles de desarrollar.
- Función:
 - Conjunto de instrucciones que realizan una determinada tarea y que devuelven un resultado al exterior. Pueden tener o no parámetros.
- Procedimiento (uso de void):
 - Lo mismo que la función pero NO devuelven nada al exterior.

Estructura de una función

```
Tipo_de_retorno nombreFunción (parámetros){  
    Cuerpo de la función  
    return Expresión  
}
```

- **Tipo_de_retorno:** Tipo de valor devuelto por la función. Podemos utilizar **void** para indicar que la función no devuelve nada → **Procedimiento**.
- **nombreFunción:** Identificador o nombre de función.
- **Lista de parámetros:** Lista de declaraciones de los parámetros de la función separados por comas.
- **Expresión:** Valor que devuelve la función.

Ejemplo de una función

```
float suma(float num1, float num2){  
    float resp;  
    resp = num1 + num2;  
    return resp;  
}
```

```
void main(void){  
    float resul = suma(5.5, 6.88);  
}
```

Aspectos sobre Funciones

- Tipo del resultado siempre se especifica delante del nombre de la función.
- Lista de parámetros: Delante del nombre del parámetro se indica el tipo y van separados por comas.
- El cuerpo de la función se encierra entre llaves.
- No se pueden declarar funciones anidadas.
- Declaración local: Constantes, tipos, variables declaradas dentro de la función se consideran locales a la misma y no perduran fuera de ella.
- Valor devuelto se indica con la palabra reservada **return**.

Paso de parámetros

- **Por valor / o copia:** Se pasa una copia de la variable, si dentro de la función se modifica, el cambio solo afecta al ámbito de la función. No afecta a la variable original.
- **Por referencia:** Se pasa la dirección de la variable, en caso de que realicemos una modificación de la misma dentro de la función afectará a la variable original.

Ejemplo paso por valor

- Por valor:

- Llamada:

- ```
float resul = suma (7.8, 6.7);
```

- Cabecera de la función:

- ```
float suma(float a, float b){ ... }
```

Ejemplo paso por referencia

- Paso por referencia:

- Llamada:

```
float x;
```

```
float y;
```

```
entrada(&x, &y); /* Extraer la dirección de las variables*/
```

- Cabecera de la función:

```
void entrada(float *a, float *b){
```

```
    *a = 0.0;
```

```
    *b = 0.0;
```

```
}
```

Parámetros const en una función

- Indican al compilador que el parámetro es de solo lectura en el interior de la función.

- Ejemplo:

```
void f1( const int x, const int *y){  
    x = 10; /* Error */  
    *y = 11; /* Error */  
    y = &x; /* Correcto */  
}
```

Recursividad

- La función recursiva es aquella que se llama a sí misma directa o indirectamente.
- Se llama así misma desde su propio cuerpo.
- En el caso de la recursividad indirecta se implica mas de una función.
- Ejemplo: `main()` llama a `uno()`, `uno` llama a `dos()`, y desde `dos()` volvemos a llamar a `uno()`.
- En un proceso recursivo siempre tenemos que tener una condición de terminación o también llamado el paso base.

Ejemplo

- La función factorial:
- $n! = n * (n-1) * (n-2) * \dots * 2 * 1$

```
int factorial(int n){  
    if (n == 1)  
        return(1);  
    else  
        return(n * factorial(n-1));  
}
```

Definición de Macros

- No forman parte del lenguaje C.
- Las gestiona el preprocesador de C.
- Deben ir al comienzo del programa.
- Todas llevan el símbolo #.
- No terminan con ;
- Se precompilan antes de las sentencias del programa.

Uso de macros

- Una **macro** es lo mismo que un texto que se sustituye en el programa fuente cuando se compila.
- Ventajas:
 - Más rápida que una función. (La ejecución).
 - Se pueden usar para cualquier tipo de dato.
 - En los parámetros de una macro se pueden meter tipos.
- Desventajas:
 - Ocupan mas que una función (se sustituye todas las veces que la llamo).
 - Cuidado con la especificación de los parámetros.

Macros sin parámetros

- `#define` identificador cadena
- identificador: es el nombre de la macro.
- cadena: será la definición de la macro, no puede haber blancos).
- Ejemplo:

```
#define uno 1
```

```
while (uno){ ... }
```

Macros con parámetros

- `#define identificador(parametros) cadena`
- Identificador: El nombre de la macro.
- Parámetros: Sin especificar el tipo y separados por comas, no podemos dejar blancos.
- Cadena: la definición de la macro, podemos utilizar los parámetros indicados.
- Ejemplo:

Ejemplos

```
#define multiplicar(a,b) a*b  
    // a y b, podrán ser de cualquier tipo.  
    // En el código: multiplicar(3,4)
```

Para intercambiar dos variables:

```
#define intercambiar(a,b) (c=a;a=b;b=c)  
// La variable c tiene que estar definida:  
int x=3, y=8, c;  
intercambiar(x,y);  
// Al sustituirse quedará así:  
c=x; x=y; y=c; // Nos sirve para varios tipos.
```

Mas operaciones con Macros

- Macros dentro de macros

```
#define uno 1
```

```
#define dos 2
```

```
#define tres ((uno)+(dos))
```

- Eliminación de una macro:

```
#undef nombre_macro
```

Ejemplo para eliminar una macro

```
#define uno 1
```

```
...
```

```
int x, y;
```

```
    x = uno;
```

```
#undef uno;
```

```
y = uno; // ERROR, ya no está definida
```


Diseño de programas

- Estructura de un programa en C.
 - declaración de importaciones
 - definición de constantes
 - definición de tipos
 - Otras funciones
 - void main (void){
 - declaración de variables
 - instrucciones ejecutables
 - }