Práctica 4. Estructuras de control iterativas y Funciones

Objetivos de la práctica:

- Adquirir la capacidad de diseñar algoritmos para la resolución de problemas básicos.
- Conocer cómo se traduce un algoritmo escrito en pseudocódigo en un lenguaje de programación.
- Adquirir la capacidad de programar en lenguaje C usando funciones.
- Conocer qué procedimiento se sigue para la declaración y uso de funciones.

1. Conceptos fundamentales

En las prácticas anteriores vimos qué etapas teníamos que seguir cuando programamos en lenguaje C:

- En papel, diseñar el algoritmo que resuelve el problema que queremos resolver.
- Traducir el algoritmo a lenguaje C. Para ello usaremos un editor de textos como por ejemplo el *gedit*.
- Compilar el programa para obtener un ejecutable. Vamos a utilizar el compilador *gcc*. El comando a utilizar sería: `$gcc -o ejecutable fuente.c`
- Ejecutar el programa y probar que su funcionamiento es el deseado. Emplearemos el comando `$./ejecutable`

2. Operaciones

En prácticas anteriores hemos visto que se puede operar con las variables:

- `a=b+c;` //Guarda en `a` la suma de `b` y `c`
- `a=b-c;` //Guarda en `a` la resta de `b` y `c`
- `a=b*c;` //Guarda en `a` el producto de `b` y `c`
- `a=b/c;` //Guarda en `a` la división de `b` y `c`
- `a=b%c;` //Guarda en `a` el resto de la división entera de `b` entre `c`

Cuando el operando es el mismo que el destino, se puede abreviar de la siguiente forma:

- `a=a+b;`
- `a+=b;`

Ambas expresiones son equivalentes.

Otras expresiones abreviadas que se pueden realizar son las siguientes:

- `a++;` // Equivale a `a=a+1;`
- `a--;` // Equivale a `a=a-1;`
- `b=a++;` // Equivale a `b=a; a=a+1;`
- `b=++a;` // Equivale a `a=a+1; b=a;`

3. Estructuras de control iterativas: bucles

Permiten definir fragmentos de un programa que se repiten (bucles). Hay que controlar que no sea un bucle infinito:

- Controlando una condición en cada iteración.
- Contando el número de iteraciones (uso de contadores).

La estructura de los bucles puede ser:

- Mientras se cumpla una condición, haz unas instrucciones.
- Haz unas instrucciones mientras se cumpla una condición.
- Haz unas instrucciones desde una condición inicial hasta que se alcance una condición final.

3.1. Bucle while

La estructura sería:

```
MIENTRAS condición HACER
 instrucciones
FINMIENTRAS
```

Y en C corresponde con:

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

Las instrucciones se ejecutan 0 o más veces.

Un ejemplo sería el siguiente:

```
int n=3;
while (n>0){
   printf("Iteración %d", n);
   n--;
}
```

Se ejecuta 3 veces el bucle, para `n` con valor 3, 2 y 1.

3.2. Bucle do-while

La estructura sería:

```
HACER
 instrucciones
MIENTRAS condición
```

Y en C corresponde con:

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

Las instrucciones se ejecutan 1 o más veces.
Un ejemplo sería el siguiente:

```
int n=3;
do{
   printf("Iteración%d", n);
   n--;
}while (n>0);
```

Se ejecuta 3 veces el bucle, para `n` con valor 3, 2 y 1.

3.3. Bucle for

La estructura sería:

```
PARA condición_inicial MIENTRAS condición HACER
 instrucciones
FINPARA
```

Y en C corresponde con:

```
for(condición_inicial; condición; actualización_contadores){
   //instrucciones
}
```

Las instrucciones se ejecutan 0 o más veces.
Un ejemplo sería el siguiente:

```
int n;
for(n=3;n>0;n--){
   printf("Iteración%d", n);
}
```

Se ejecuta 3 veces el bucle, para `n` con valor 3, 2 y 1.

4. Algoritmos

4.1. Algoritmo Suma_n_primeros_nums_v1

El algoritmo siguiente calcula la suma de los n primeros números enteros:

```
ALGORITMO Suma_n_primeros_nums_v1
 ENTRADAS:
  Num: Entero ; Números a sumar
 SALIDAS:
  Total: Entero ; Suma de los n primeros números
```

```
VARIABLES:
 Num: Entero
 Total: Entero
 i: Entero ; Contador
INICIO
 ESCRIBA "Escribe cuantos números quieres sumar: "
 LEA Num
 SI Num ≥ 1 ENTONCES
   Total ← 0
   i ← 1
   MIENTRAS i ≤ Num HACER
     Total ← Total + i
     i ← i + 1
   FINMIENTRAS
   ESCRIBA "La suma es: "
   ESCRIBA Total
 SINO
   ESCRIBA "El número ha de ser mayor o igual a 1"
 FINSI
FIN
```

4.2. Algoritmo Potencia

El algoritmo siguiente calcula la potencia de un número elevado a un exponente:

```
ALGORITMO Potencia
 ENTRADAS:
  Base: Entero ; Número leído (base)
  Exp: Entero ; Número leído (exponente)
 SALIDAS:
  Pot: Entero ; Potencia (Base elevado a Exp)
 VARIABLES:
  Base: Entero
  Exp: Entero
  Pot: Entero
  i: Entero
 INICIO
  ESCRIBA "Escribe un número (base): "
  LEA Base
  ESCRIBA "Escribe un número (exponente): "
  LEA Exp
  SI Exp ≥ 1 Y Base ≥ 1 ENTONCES
    Pot ← 1
    i ← 1
    MIENTRAS Exp ≥ 1 HACER
      Pot ← Pot * Base
      Exp ← Exp - 1
    FINMIENTRAS
    ESCRIBA "La potencia es: "
    ESCRIBA Pot
```

```
    SINO
      ESCRIBA "La base y el exponente han de ser mayores o iguales
a 1"
    FINSI
   FIN
```

5. Traducción a C

5.1. Programa Suma_n_primeros_nums_v1

Si traducimos el algoritmo Suma_n_primeros_nums_v1 a lenguaje C nos quedaría el programa siguiente:

```
//Librería que contiene las funciones scanf y printf
#include <stdio.h>

//Función principal del programa
int main ()
{
  // Este programa calcula la suma de los primeros n números

  // Declaro las variables de mi función
  int Num, Total, i;

  //Sustituyo la función ESCRIBA "cadena" por printf
  printf("Escribe cuantos números quieres sumar: ");

  //Sustituyo la función LEA Num por scanf ("%d", &variableEntera);
  scanf("%d", &Num); //Guarda el número leído en la variable Num


  //Inicializo las variables
  Total = 0;
  i = 1;

  //Compruebo si el número introducido es mayor o igual que 1
  if (Num >= 1) {
    //Hago el bucle para sumar hasta que i valga n
    while (i <= Num){
      //Calculo la suma parcial
      Total = Total + i;
      i = i + 1;
    }

    //Sustituyo la función ESCRIBA "cadena" por printf
    printf("La suma es: %d \n", Total);
  }
  else{
    //Sustituyo la función ESCRIBA "cadena" por printf
    printf("El número ha de ser mayor o igual a 1");
```

```
    }
    //Fin del programa
    return 0;
  }
```

5.2. Programa Potencia

La traducción del algoritmo Potencia a lenguaje C sería la siguiente:

```
//Librería que contiene las funciones scanf y printf
#include <stdio.h>

//Función principal del programa
int main ()
{
  // Este programa calcula la potencia de un número

  // Declaro las variables de mi función
  int Base, Exp, Pot, i;

  //Sustituyo la función ESCRIBA "cadena" por printf
  printf("Escribe un número (base): ");

  //Sustituyo la función LEA Base por scanf ("%d", &variableEntera);
  scanf("%d", &Base); //Guarda el número leído en la variable Base


  //Sustituyo la función ESCRIBA "cadena" por printf
  printf("Escribe un número (exponente): ");

  //Sustituyo la función LEA Exp por scanf ("%d", &variableEntera);
  scanf("%d", &Exp); //Guarda el número leído en la variable Exp


  //Compruebo que la base y el exponente sean mayores que 1
  if ((Base >= 1) && (Exp >= 1)){
    //Inicializo las variables
    Pot = 1;
    i = 1;
    while (Exp >= 1){
      Pot = Pot * Base;
      Exp = Exp - 1;
    }
    //Sustituyo la función ESCRIBA por printf
    printf("La potencia es: %d", Pot);
  }
  else{
    //Sustituyo la función ESCRIBA "cadena" por printf
    printf("La base y el exponente han de ser mayores o iguales
a 1");
```

```
    }

    //Fin del programa
    return 0;
}
```

6. Las funciones

Las funciones son fragmentos independientes de código fuente diseñados para realizar una tarea específica.

Sus ventajas son:

- Facilita la programación: permite descomponer un programa en bloques.
- Mejora la legibilidad del código.
- Permite reutilizar código.
- etc

En C todo son funciones.

6.1. Función main

La función main es la función principal del programa. Cuando ejecutamos un programa con `$./ejecutable` la primera instrucción que se ejecuta es la primera instrucción de la función `main`.

6.2. Sintaxis

La manera de definir una función es la siguiente:

```
tipo_datos_resultado nombrefuncion(tipo_datos_argumento nombreargumento1)
{

  //Aquí van las instrucciones de la función
  //Todas acaban en ;

  //La última instrucción es un return
}
```

- El nombre de la función es el que queramos, sin espacios, sin acentos, etc.
- Los argumentos son los datos que se le pasan a la función. Van entre paréntesis. Para cada argumento hay que indicar el tipo de datos y el nombre. Se separan por comas. Ejemplo: `(int base, int altura)`
- `tipo_datos_resultado` es el tipo de datos del resultado que devuelve la función. Se pone delante del nombre de la función. Si no devuelve nada se indica `void`, si devuelve un entero, `int`, etc.
- Las instrucciones de la función van entre llaves.
- Para devolver un resultado se usa la instrucción `return`

6.3. Estructura de un programa con funciones

Usando funciones, la estructura de un programa tiene la declaración de prototipos, antes del main, y la definición de las funciones después del main. Los prototipos tienen la misma línea que la definición pero acaba en punto y coma. Por ejemplo `tipo_datos_resultado nombrefuncion(tipo_datos_argumento nombreargumento1);`

Un esquema sería la siguiente:

```
//Librerías que usa el programa C
//SINTAXIS: #include <nombrelibrería>
...
//Definición de constantes, etc
//SINTAXIS: #define NOMBRECONSTANTE <valor_constante>
...

//Declaración de prototipos de funciones
tipo_datos_resultado nombrefuncion1(tipo_datos_argumento nombreargumento1);
tipo_datos_resultado2 nombrefuncion2(tipo_datos_argumento1 nombreargumento1,
tipo_datos_argumento2 nombreargumento2);


//Función principal del programa
int main ()
{

  //Aquí van las instrucciones de nuestro programa.
  //Todas acaban en ;

  return 0;
}

//Resto de funciones del programa
tipo_datos_resultado nombrefuncion1(tipo_datos_argumento nombreargumento1)
{

  //Aquí van las instrucciones de la función
  //Todas acaban en ;

  //La última instrucción es un return
}

tipo_datos_resultado2 nombrefuncion2(tipo_datos_argumento1 nombreargumento1,
tipo_datos_argumento2 nombreargumento2)
{

  //Aquí van las instrucciones de la función
  //Todas acaban en ;

  //La última instrucción es un return
}
```

6.4. Llamadas a funciones

Una vez que hemos declarado los prototipos y definido las funciones, podemos llamarlas desde cualquier parte del código. Para ello, se indica el nombre de la función y entre paréntesis los valores/variables que le pasamos como argumentos. Podemos tener varios casos, que no tengan argumentos, que tengan uno, que tengan más de uno. Además, hay funciones que no devuelven nada (`void`) y otras que sí. Las que devuelven algo necesitan almacenar ese valor en una variable (o bien se evalúa en una condición. Por ejemplo `if (funcion()>0){...}`).

Por ejemplo:

```
int main ()
{

  int valor, a, b;
  //Función sin argumentos que devuelve un int
  valor=funcion1();
  //Función con dos argumentos enteros que devuelve un int
  valor=funcion1(1, 5);
  //Función con dos argumentos enteros que devuelve un int
  valor=funcion1(a, b);
  //Función con dos argumentos enteros que no devuelve nada
  funcion1(a, b);
  //Función sin argumentos que no devuelve nada
  funcion1();
  return 0;
}
```

6.5. Ejemplo

El programa siguiente tiene una función que calcula el área de un triángulo.

```
#include <stdio.h>
float areaTriangulo(float base, float altura);
int main () {
  float b, a, area;
  printf("Introduzca la base del triángulo");
  scanf("%f", &b);
  printf("Introduzca la altura del triángulo");
  scanf("%f", &a);
  //Llamada a la función. Deja el resultado en area
  area=areaTriangulo(b,a);
  printf("El área es%f", area);
  return 0;
}
float areaTriangulo(float base, float altura){
  float resultado;
  resultado=base*altura/2;
  return (resultado);
}
```

7. Ejercicios propuestos

7.1. Ejercicio 1

– Escribe en un archivo con extensión `.c` el programa `Suma_n_primeros_nums_v1.` Compílalo con el compilador `gcc`. Después, ejecútalo para comprobar su funcionamiento. ¿Cómo sería el código usando una estructura de tipo *do-while*? ¿Y un bucle *for*?

7.2. Ejercicio 2

– Escribe en un archivo con extensión `.c` el programa `Potencia`. Compílalo con el compilador `gcc`. Después, ejecútalo para comprobar su funcionamiento. ¿Cómo sería el código usando una estructura de tipo *do-while*? ¿Y un bucle *for*?

7.3. Ejercicio 3

– Realice un programa `volumen.c` en lenguaje C que calcule volúmenes de figuras geométricas.

- La función `main` presentará un menú para saber si se quiere calcular el volumen de un cono (1) o el volumen de ortoedro (2). En función de la opción elegida, se leerán los datos necesarios y se llamará a la función `volumen_cono` o a la función `volumen_ortoedro`. La función `main` imprimirá el resultado que le devuelva la función.
- La función `volumen_cono` recibe como argumentos el radio de la base y la altura. Devuelve el volumen del cono $(1/3 * PI * radio^2 * altura)$
- La función `volumen_ortoedro` recibe como argumentos dos lados de la base y la altura. Devuelve el volumen del ortoedro $(lado1 * lado2 * altura)$

7.4. Ejercicio 4

Modifica el programa anterior para que el menú incluya una opción 3 "Salir". Se mostrará el menú y mientras la opción elegida no sea la 3 no se saldrá del programa. Es decir, se muestra el menú, si la opción es 1 ó 2 se calcula el volumen, se imprime por pantalla y vuelve a mostrar el menú. Si la opción es distinta de 1, 2 y 3 da un mensaje de error y vuelve a mostrar el menú. Si la opción es 3, sale del programa.

7.5. Ejercicio 5

Realice un programa en C que lea dos números y compruebe que el primero es menor que el segundo. Si no es así dará un mensaje de error. Si los números son correctos, llamará a una función que se le pasan los dos números e imprime todos los números pares desde el mayor hasta el menor por pantalla. Por ejemplo si se lee el 3 y el 7, ha de imprimir 6, 4. Compílalo con el compilador `gcc`. Después, ejecútalo para comprobar que funciona correctamente.

7.6. Ejercicio 6

Realice un programa en C que lea dos números y compruebe que son positivos. Si no lo son debe dar un mensaje de error. Si son positivos, debe llamar a una función que dados dos números, base y exponente, calcula la base elevada al exponente. La función devuelve el resultado y el main lo imprime por pantalla. Compílalo con el compilador `gcc`. Después, ejecútalo para comprobar que funciona correctamente.