CFIGM SISTEMAS MICROINFORMÁTICOS Y REDES

Sistemas Operativos Monopuesto

TEMA 03: Fundamentos de Programación I

Introducción a la programación: Lenguaje C

Contenido

- Introducción
- Mi primer programa
- Etapas a seguir en la programación
- Estructura de un programa
- Elementos léxicos de un programa.
- Entrada/Salida
- Ejercicios

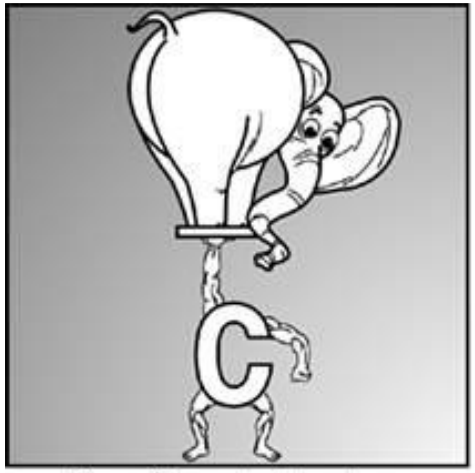
Introducción

- ¿Qué es un algoritmo?
  "Conjunto finito de instrucciones para resolver un problema"

- ¿Qué es un programa?
  "Un algoritmo que ejecuta un ordenador".

  El algoritmo debe estar escrito en un lenguaje de programación

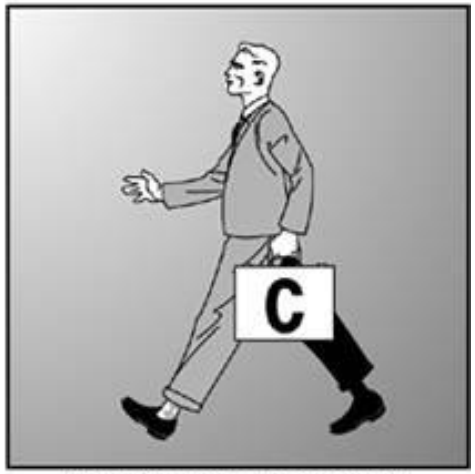
Introducción: características lenguaje C



Powerful control structures

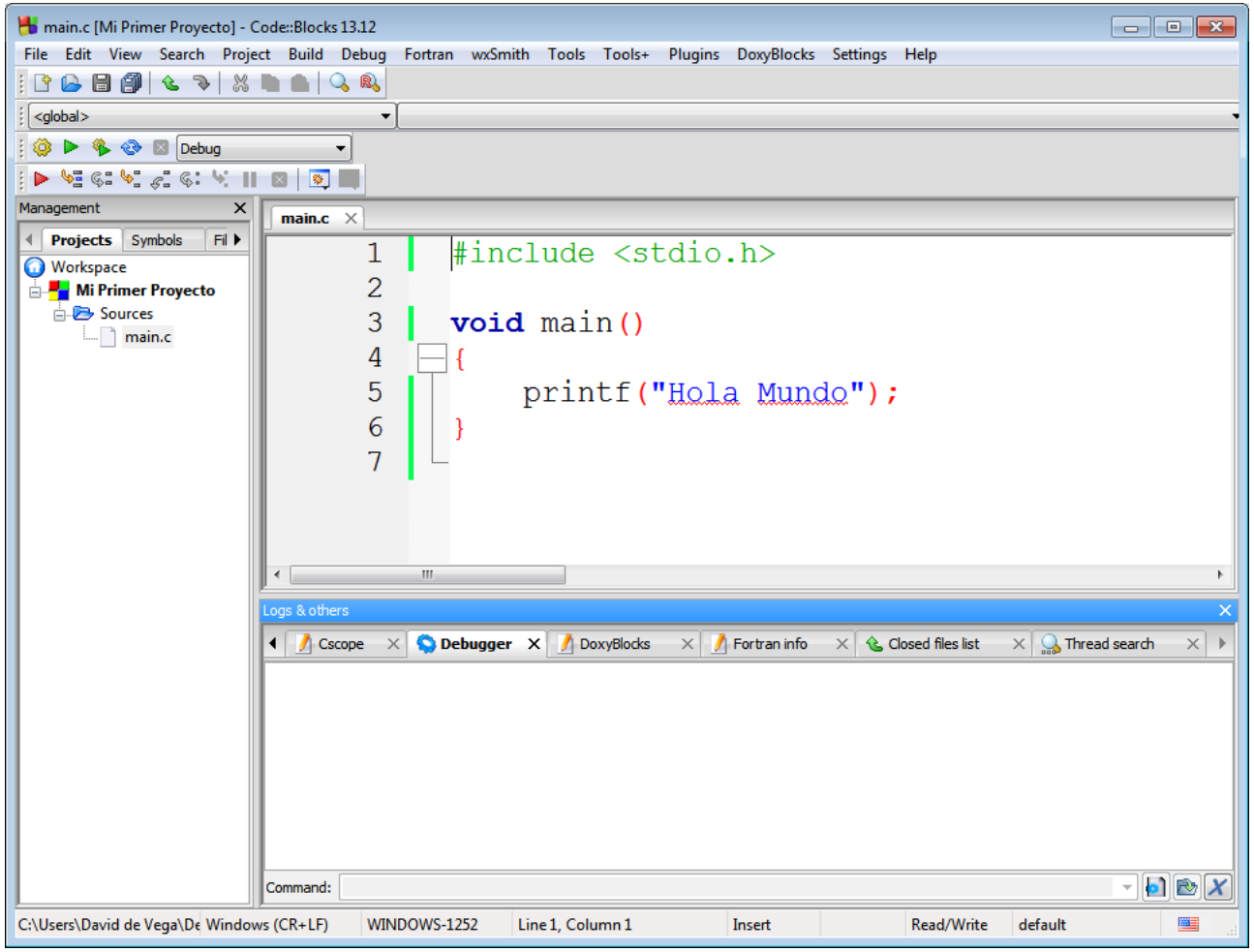

Fast


Compact code — small programs



Portable to other computers

Mi primer programa



Etapas a seguir en la programación

1. **Edición**
2. **Compilación**
3. **Enlazado**
4. **Ejecución**
5. **Depuración →Hacer una traza**

COMPETENCIA: Hacer una traza

Etapas a seguir en la programación

1. **Edición**
   - Escribimos las instrucciones y lo almacenamos en un archivo con la extensión '.c'. Es lo que llamamos el código **fuente**.
2. **Compilación**
3. **Enlazado**
4. **Ejecución**
5. **Depuración**

Etapas a seguir en la programación

1. **Edición**
2. **Compilación**
   - Para poder ser ejecutado, la secuencia de instrucciones del programa debe ser convertida a un código comprensible para la máquina. Para eso es necesario un traductor (el **compilador**) que lee el archivo fuente y genera el archivo **objeto** (habitualmente con la extensión .obj). Los errores de esta fase se llaman errores de compilación.
3. **Enlazado**
4. **Ejecución**
5. **Depuración**

Etapas a seguir en la programación

1. **Edición**
2. **Compilación**
3. **Enlazado**
   - En la fase de enlazado, a partir del archivo objeto se genera el archivo **ejecutable** (extensión .exe).
4. **Ejecución**
5. **Depuración**

Etapas a seguir en la programación

1. **Edición**
2. **Compilación**
3. **Enlazado**
4. **Ejecución**
   - Finalmente, podemos ejecutar nuestro programa en la fase de ejecución. Los errores de esta fase se llaman errores de ejecución.
5. **Depuración**

Etapas a seguir en la programación

1. **Edición**
2. **Compilación**
3. **Enlazado**
4. **Ejecución**
5. **Depuración**

   Seguir instrucción a instrucción analizando el valor de las variables

Estructura de un programa

```
#include <stdio.h>
Definición de constantes
void main() {
   definición de variables;
   instrucciones;
}
```

¿Qué es #include <stdio.h>?

stdio →Inglés “**st**an**d**ard **i**nput/**o**utput”

Se utiliza para comunicar al compilador dónde está la información necesaria para que el programa lea/escriba datos

Ejemplo 1

Escribir un programa que escriba en la pantalla “Hola a todos los alumnos de 1º SMR”

Elementos léxicos de un programa

- Identificadores
- Palabras reservadas
- Operadores y separadores
- Comentarios
- Tipos de datos
- Constantes y variables

Identificadores

- Un identificador es una secuencia de caracteres, letras, dígitos y subrayados.
- El primer carácter debe ser una letra o un subrayado.
- Las mayúsculas y minúsculas son diferentes.
- Ejemplos:
  - precio
  - precioTotal
  - sueldo_minimo
  - valor_articulo_34
  - Precio

Palabras reservadas

- Las palabras reservadas tienen un significado específico y únicamente se pueden utilizar con ese significado en un programa.

double int struct break else long switch case enum typedef char return const short unsigned continue for signed void default sizeof do if while

- Por ejemplo, no se puede crear una variable que se llame **while**

Operadores y separadores

- Todas las sentencias deben terminar con ;

- Operadores aritméticos: + - * /
  - a+b
  - a-b

- En general:

  ! % ^ & * ( ) - + = { } ~ [ ] \ ; ’ : < > ? , . / ”

Operadores aritméticos

Operador	Tipos enteros	Tipos reales
+	Suma	Suma
-	Resta	Resta
*	Producto	Producto
/	Cociente	División
%	Resto	
++	Incremento	Incremento
--	Decremento	Decremento

Operadores relacionales

Operador	Significado
>	Mayor estricto que
>=	Mayor o igual que
==	Igual a
<=	Menor o igual que
<	Menor estricto que
!=	Diferente a

Operadores lógicos

Operador	Significado
&&	Y
||	O
!	NO

- La evaluación de operaciones lógicas ocurre de izquierda a derecha y se detiene tan pronto como es posible.

¿Qué resulta de?

- **3*4+5** -> Reglas de **prioridad**. Establecen prioridad entre los operadores. Los paréntesis pueden alterar el orden de ejecución de las operaciones, ya que las expresiones que están dentro de los paréntesis se evalúan en primer lugar.
- **6/2/3** -> Reglas de **asociatividad**. Resuelven la ambigüedad entre operadores de la misma prioridad
- Siempre podéis utilizar paréntesis en vuestros programas para evitar estos casos en los que se aplica la prioridad y la asociatividad si no lo tenéis claro.
- Debéis conocer las reglas para poder entender los programas que escriben otros.

Prioridad y asociatividad

Operadores (ordenados de arriba a abajo de mayor a menor prioridad)	**Asociatividad**
!, ++, --, - (unario)	De derecha a izquierda
*, /, %	De izquierda a derecha
+, -	De izquierda a derecha
<, <=, > , >=	De izquierda a derecha
==, !=	De izquierda a derecha
&&	De izquierda a derecha
||	De izquierda a derecha

Comentarios

- Se marca el inicio de un comentario con ***/**** y su finalización con ****/***.
- Desde el año 99, se pueden utilizar comentarios “en línea”. Éstos comienzan con ***//*** y se alargan hasta el final de la línea.
- Ejemplos:
  - /* Guardamos en fichero */
  - x++; // Incrementamos el valor de x

Tipos de datos

- C nos ofrece diferentes tipos fundamentales:
  - char: caracteres [‘a’,’x’,’%’,’\n’]
  - int, long: números enteros [-34,13,1982]
  - float, double: números con decimales (punto flotante) [0.23,-1.22,65454.343]

Constantes simbólicas

La sintaxis es:

```
#define identificador_constante  valor_constante
```

Pueden ser de varios tipos de datos. En el caso de constantes de tipo carácter, éstas contienen un único carácter entre comillas simples, en el caso de una constante de tipo cadena, éstas contienen una secuencia de caracteres entre comillas dobles.

Ejemplos:

```
#define PI 3.14159
#define MAXIMO 999
#define ULTIMALETRA 'Z'
#define MENSAJE "Introduzca su edad:“
```

En la compilación se sustituyen los identificadores (PI, MAXIMO, ULTIMALETRA, MENSAJE) con su valor real (3.14159, 999,’Z’, “Introduzca su edad”).

Variables

- Una variable es una posición de memoria donde se almacena un valor de un cierto tipo de dato.
- Las variables tienen un nombre que describe su propósito.
- Su declaración debe aparecer al principio de un bloque (justo detrás de una { o de otra declaración).

Declaración de variables

- La declaración tiene siempre la estructura

  tipo nombre_variable;

  tipo es un tipo de dato conocido en C

  nombre_variable es un identificador válido en C

- Ejemplos:
  - long numHoras;
  - double anguloRotacion;
  - float NotaMedia;
  - char c;

Inicialización de variables

- Se puede proporcionar un valor a una variable en el momento de su declaración.

  tipo nombre_variable = expresion;

  dónde *expresion* es cualquier expresión válida del mismo tipo del que se define la variable
- Ejemplos:
  - char respuesta = ‘S’;
  - int contador = 1;
  - float peso = 87.3*56.9;
  - int anyo = 2008;

Asignación de valores a variables

- Se puede modificar el valor de una variable en cualquier momento del programa.

  nombre_variable = expresion;

  dónde *expresion* es cualquier expresión válida del mismo tipo de la variable

- Ejemplos:
  - x = (z-3)*12;
  - segundos = minutos*60;
  - f = m*a
  - en_pesetas = en_euros * 166.386;

Ejemplo 2

Escribir un programa que calcule el área de un círculo de radio 3.

Entrada/Salida

Las funciones de entrada salida se encuentran en la librería estándar <stdio.h>.

Leer → *scanf*

Escribir →*printf*

Escribir

La instrucción **printf** permite escribir una lista de datos con un formato preestablecido. Acepta diferentes tipos de argumentos: carácter, valor numérico entero o real o cadena de caracteres, y los escribe según un formato especificado sobre la salida estándar. La forma de la función printf es:

printf ("formato", arg1, arg2, ..., argn);

argi pueden ser constantes, variables o expresiones **“formato”** es una serie de caracteres en la cual se pueden encontrar dos tipos de datos: texto a escribir literalmente, y los símbolos especificadores del formato con el cual se van a escribir las variables dadas como argumentos.

Especificadores de formato para printf

Especificador	Tipo de datos
%c	Carácter
%d	Número entero
%f	Número real (float)
%lf	Numero real (double)
%s	Cadena de caracteres

Ejemplos de printf

1) float x=23.32;
   printf (“\nEl cuadrado de %f vale %f”, x, x*x);

2) int edad = 34;
   printf (“\nLa edad de %s es %d años”, “Juan”, edad);

3) double x = 3.141592654;
   printf (“\nSeno (%f) = %f\n”, x, sin(x));

Leer

La instrucción **scanf** permite leer valores desde la entrada estándar y almacenarlos en las variables que se especifican como argumentos. Éstas deben ir normalmente precedidas por el símbolo &. La sintaxis es:

scanf ("formato", arg1, arg2, ..., argn);

donde debe haber tantos especificadores en la cadena de **“formato”** como variables en la lista de argumentos. Los especificadores de formaqto son los mismos que para la función **printf**

Ejemplos de scanf

1) int horas;
   printf(“Introduzca el numero de horas\n”);
   scanf(“%d”,&horas);

2) char letra;
   printf(“Introduzca una letra\n”);
   scanf(“%d”,&horas);

3) int horas; double velocidad_media;
   printf(“Introduzca las horas conduciendo y la vel. media”);
   scanf(“%d%lf”,&horas,&velocidad_media);

Ejemplo 3

Escribir un programa que calcule el área de un círculo cuyo radio r sea leído desde teclado.