Programación en C

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Índice General

- Fundamentos de lenguaje C.
- Tipos de datos, operadores y expresiones.
- Instrucciones de entrada/salida.
- Instrucciones de control.
- Arrays.
- Estructuras y uniones.
- Punteros.
- Funciones y organización de programas.
- Ficheros.
- Estructuras de datos.

Fundamentos de lenguaje C

Índice

- Lenguajes de Programación.
- Clasificación de los Lenguajes de Programación.
- Lenguajes de Programación de alto nivel.
- Historia del lenguaje C.
- Características de C.
- Proceso de programación.
- Entornos de programación.
- Estructura de un programa C.
- Ejemplos de programas comentados.
- Buenas prácticas de programación.
- Errores comunes de programación.

Lenguajes de Programación

- Un lenguaje de programación es un lenguaje artificial capaz de ser leído por una máquina diseñado para *especificar* de manera precisa las *acciones* que deben ser realizadas por un computador sobre *datos* almacenados en la misma.
- Un lenguaje de programación consta de un conjunto de *símbolos* y *reglas sintácticas y semánticas* que definen su estructura y el significado de sus elementos y expresiones.

Clasificación de los LP

- Los lenguajes de programación se pueden clasificar atendiendo a varios criterios:
  - Según el nivel de abstracción
  - Según la forma de ejecución
  - Según el paradigma de programación que poseen cada uno de ellos

Clases de LP por nivel de abstracción (I)

- Lenguajes de máquina
  - lenguajes legibles directamente por la computadora, consistentes en cadenas binarias que especifican instrucciones y datos.
  - Ejemplo: MIPS 32 bits

```
100011 00011 01000 00000 00001 000100   - LOAD
000010 00000 00000 00000 00100 000000   - JUMP
```

Clases de LP por nivel de abstracción (II)

- Lenguajes de bajo nivel (ensamblador)
  - Son lenguajes de programación que se acercan al funcionamiento de una computadora.
  - El más representativo es el lenguaje ensamblador que permite escribir programas en código máquina, específico para cada arquitectura de computadoras, legible por un programador.
  - Ejemplo:

```
.model small
.stack
.data
Cadena1 DB 'Hola Mundo.$'
.code

programa:
   mov ax, @data
   mov ds, ax
   mov dx, offset Cadena1
   mov ah, 9
   int 21h
end programa
```

Clases de LP por nivel de abstracción (III)

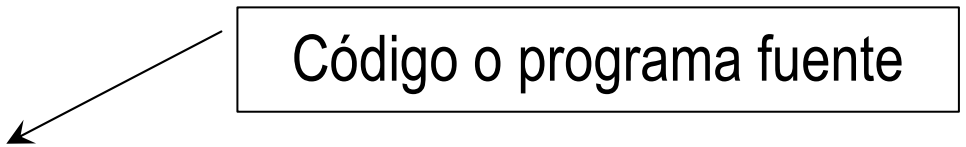
- Lenguajes de medio nivel
  - Son lenguajes de programación que tienen ciertas características que los acercan a los lenguajes de bajo nivel pero teniendo, al mismo tiempo, ciertas cualidades que lo hacen un lenguaje más cercano al humano y, por tanto, de alto nivel. El más representativo es el C.
  - Ejemplo:

```
#include <stdio.h>

int main(void)
{
        printf("Hola mundo!\n");
        return(0);
}
```

Clases de LP por nivel de abstracción (IV)

- Lenguajes de alto nivel
  - Son lenguajes de programación que están formados por elementos de lenguajes naturales, como el inglés.
  - Para los cálculos usan notaciones matemáticas.
  - Ejemplo:

```
#include <stdio.h>

main()
{
 int fahr; double celsius;

 for(fahr = 0; fahr <= 100 ;fahr += 10) {
  celsius = (5.0/9.0)*(fahr - 32.0);
  printf("Fahrenheit = %3d\tCelsius = %6.1f\t",fahr, celsius);
 }
 return 0;
}
```



Clases de LP por la forma de ejecución

- Lenguajes compilados
  - Un programa escrito en un lenguaje de alto nivel tiene que traducirse a código máquina mediante un ***compilador.*** Los ***compiladores*** son programas que traducen un programa escrito en un determinado lenguaje de programación a lenguaje máquina.
- Lenguajes interpretados
  - Utilizan un programa **intérprete** que convierte cada sentencia del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el procesamiento de los datos.

Clases de LP según el paradigma de programación

- Un **paradigma de programación** representa un enfoque particular o filosofía para la construcción del software.
- Atendiendo al paradigma de programación, se pueden clasificar los lenguajes en :
  - Imperativo o procedimental. Ej. C.
  - Funcional. Ej. LISP.
  - Lógico. Ej. PROLOG.
  - Orientado a objetos. Ej. Smalltalk, Java.

Lenguajes de Programación de alto nivel

- Hay decenas de lenguajes de programación de alto nivel. Los más representativos (aparte del **C**) son:
  - **Fortran**, desarrollado por IBM en los 1950 para uso de aplicaciones científicas y de ingeniería que requieren cálculos complejos
  - **Cobol**, desarrollado en 1950 para uso en aplicaciones comerciales.
  - **Pascal**, desarrollado por N. Wirth en 1971 para la enseñanza de la programación estructurada.
  - **Ada**, desarrollado en la década de 1970-80 para soportar multitarea.

Lenguajes de Programación de alto nivel

- Otros lenguajes con sintaxis similar a **C**:
  - **C++**, superconjunto de C desarrollado por Bjarne Stroustrup en los Laboratorios Bell. Es orientado a objeto.
  - **Java**, desarrollado por James Gosling en Sun. Capacidad de crear páginas Web dinámicas e interactivas. Utilizado en dispositivos de consumo (móviles, domótica, etc.)
  - **C#**, desarrollado por Microsoft para la plataforma .Net.

Historia del lenguaje C (I)

- Raíces: lenguaje BCPL (**1967**) y el lenguaje B (Thompson).
- **1972**: D. Ritchie diseña el lenguaje C y lo implementa en una PDP-11.
- **1974**: UNIX reescrito en C.
- C gana popularidad. Distribución gratuita de UNIX y compiladores C a las universidades.
- **1978**: Aparece el libro de Kernighan & Ritchie. La versión C se conoce como **K&R** C (Tradicional C).

Historia del lenguaje C (II)

- Comienzos de los 80: aparecen varios dialectos de C. Esfuerzo para desarrollar un C standard. Resultado: **ANSI C**.
- **1988**: 2ª edición del libro de Kernighan & Ritchie. **C standard**.
- **1999**: Actualización de la norma. Resultado: **C99**.

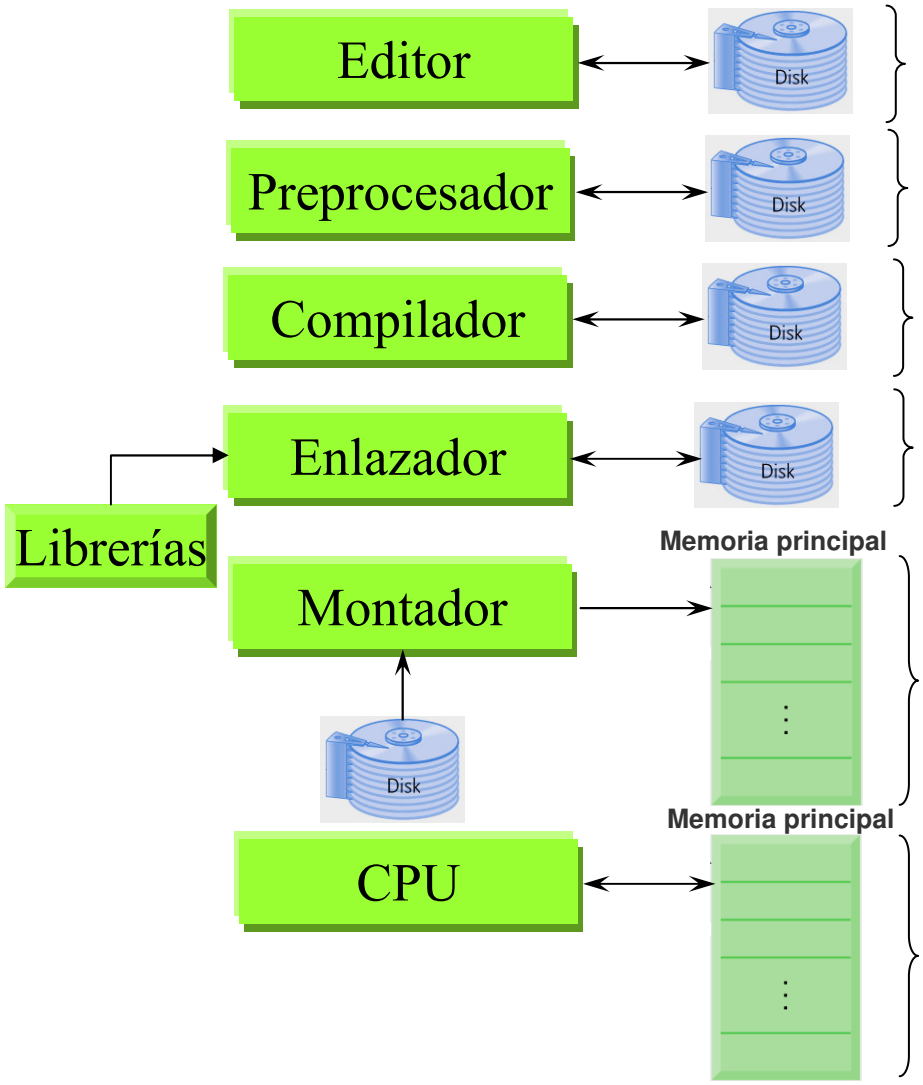
Características de C

- Promueve la programación estructurada (amplio conjunto de operadores y un conjunto completo de instrucciones).
- Diferentes tamaños de tipos de enteros y reales.
- Declaraciones de punteros a variables y funciones.
- **Preprocesador** antes de la compilación.
- Compilación de las funciones de un programa por separado.
- Más flexibilidad usando **librerías** de funciones.
- **Portabilidad**: C es un lenguaje independiente del hardware. Quiere decir que un código en C puede correr con ninguna o pocas modificaciones en un amplio rango de sistemas de computación.

Proceso de Programación

- ***Edición*** de los ficheros fuente (.c).
- ***Preprocesado*** de los ficheros fuente.
- ***Compilación*** de los ficheros fuente. Produce ficheros objetos (.obj).
- ***Enlazado*** (Linkado) de los ficheros objeto y librerías (.lib) de funciones. Produce ficheros ejecutables (.exe).
- ***Carga y Ejecución*** del código máquina.
- ***Prueba*** y ***depuración*** (debug) del programa ejecutable.

Proceso de Programación



Fase 1:
El programador crea el programa en el **editor** y lo almacena en el disco (código fuente .c).

Fase 2:
El programa **preprocesador** procesa el código.

Fase 3:
El **compilador** crea el código objeto y lo almacena en el disco (.obj).

Fase 4:
El **enlazador** enlaza el código objeto con las librerías, crea el fichero ejecutable (.exe) y lo almacena en el disco.

Fase 5:
El **montador** o cargador lee el fichero ejecutable y lo coloca en la memoria.

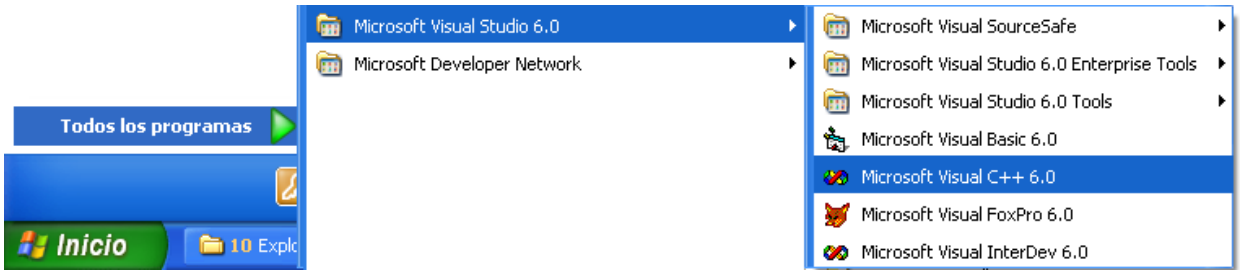
Fase 6:
La CPU toma cada instrucción y la ejecuta. Es posible que almacene nuevos datos según el programa se ejecuta.

Entornos de Programación

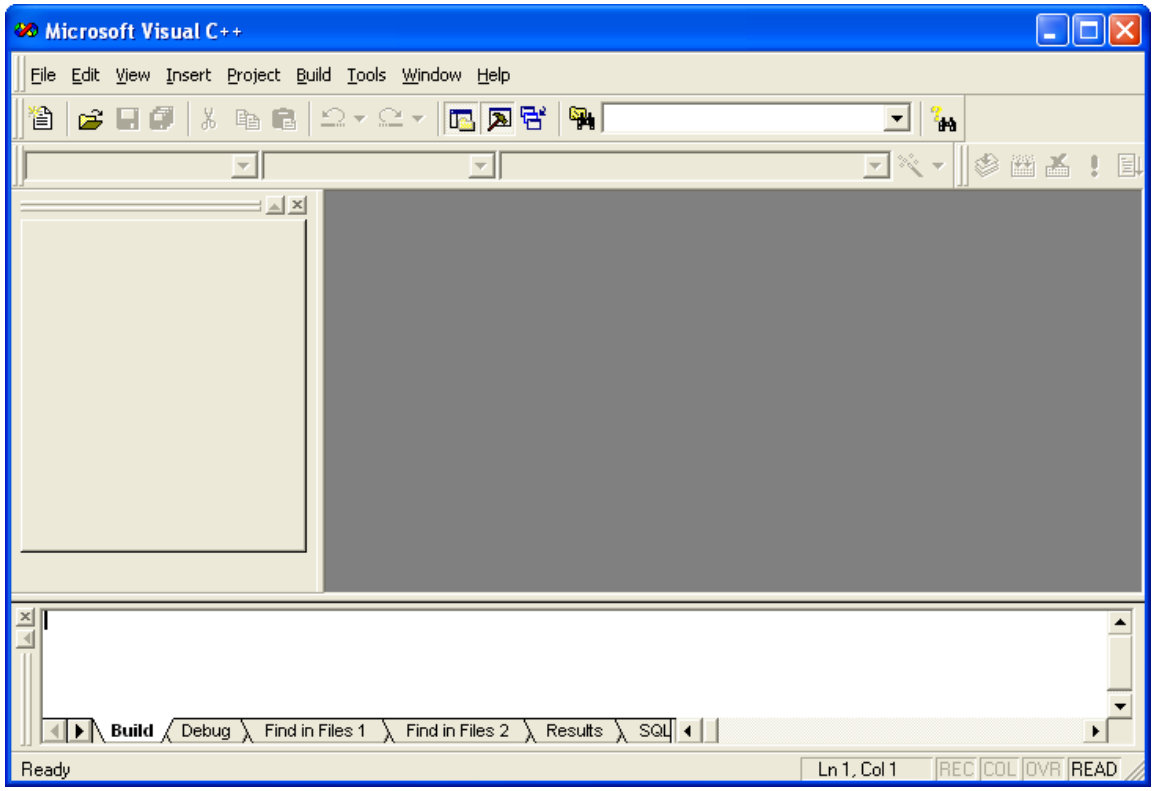
- Existen numerosos entornos de programación para el lenguaje C.
- Usaremos **Microsoft Visual Studio 6** que se puede encontrar en el software distribuido por el Servicio de Informática (http://www.unican.es/WebUC/Unidades/SdeI/).
- MS Visual Studio es un **entorno integrado** de desarrollo de programas en lenguajes C/C++. Es decir integra en un único programa todas las fases del proceso de programación.
- Es necesario crear un proyecto donde colocamos los ficheros fuentes del programa a desarrollar.

MS Visual Studio 6 - Inicio

- Iniciar MSVS:

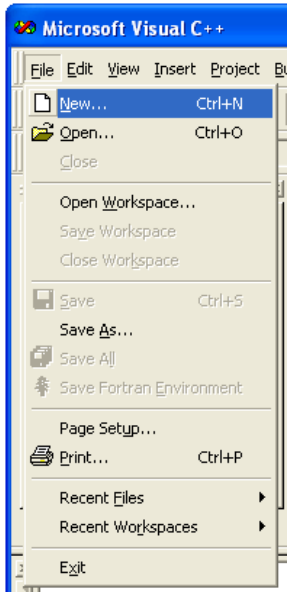


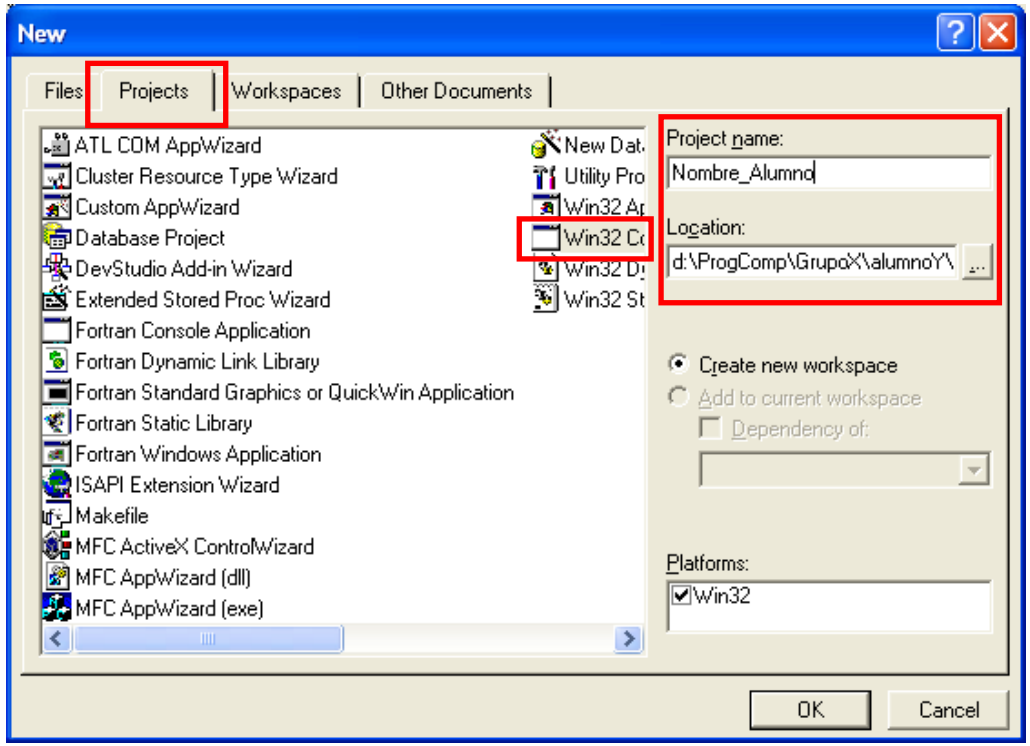
- Entorno:

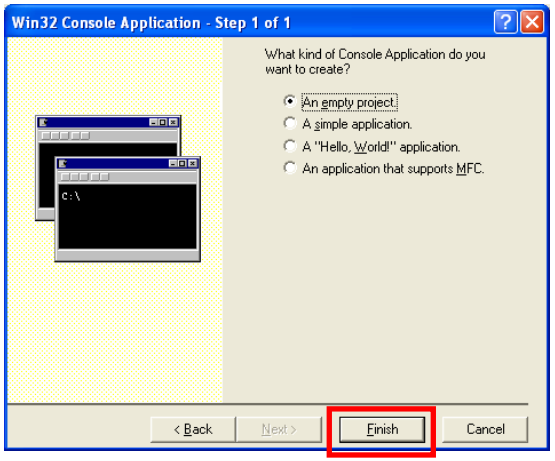


MS Visual Studio 6 – Crear proyecto

- Crear proyecto:
  - Pestaña: Projects – Tipo: Win32 Console Application
  - Nombre Proyecto (Project name): Nombre_Alumno
  - Ubicación (Location): d:\ProgComp\GrupoX

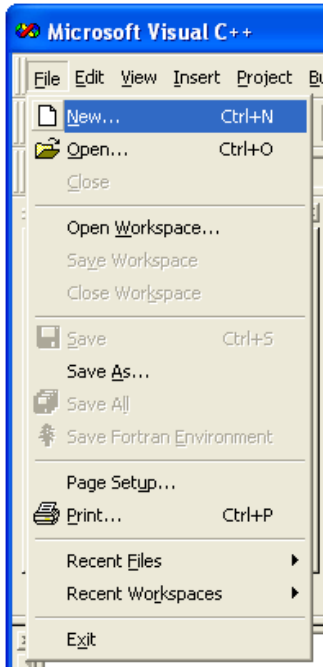


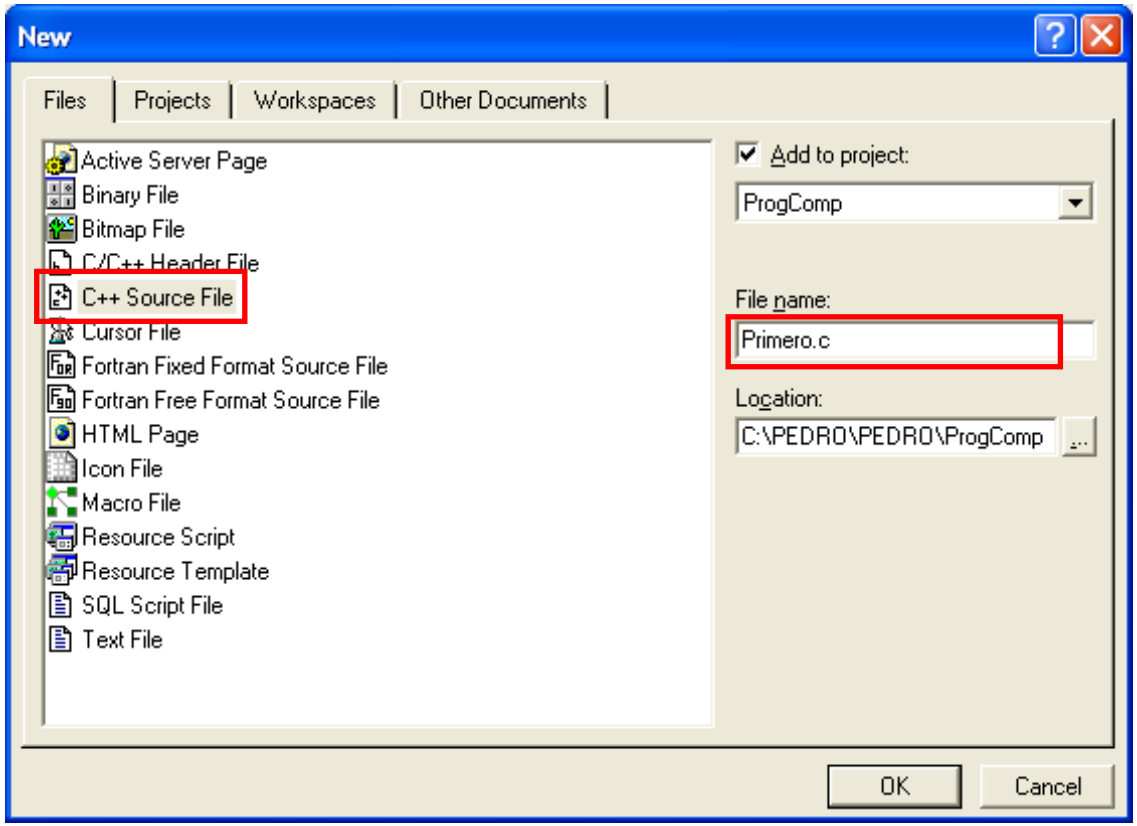




MS Visual Studio 6 - Edición

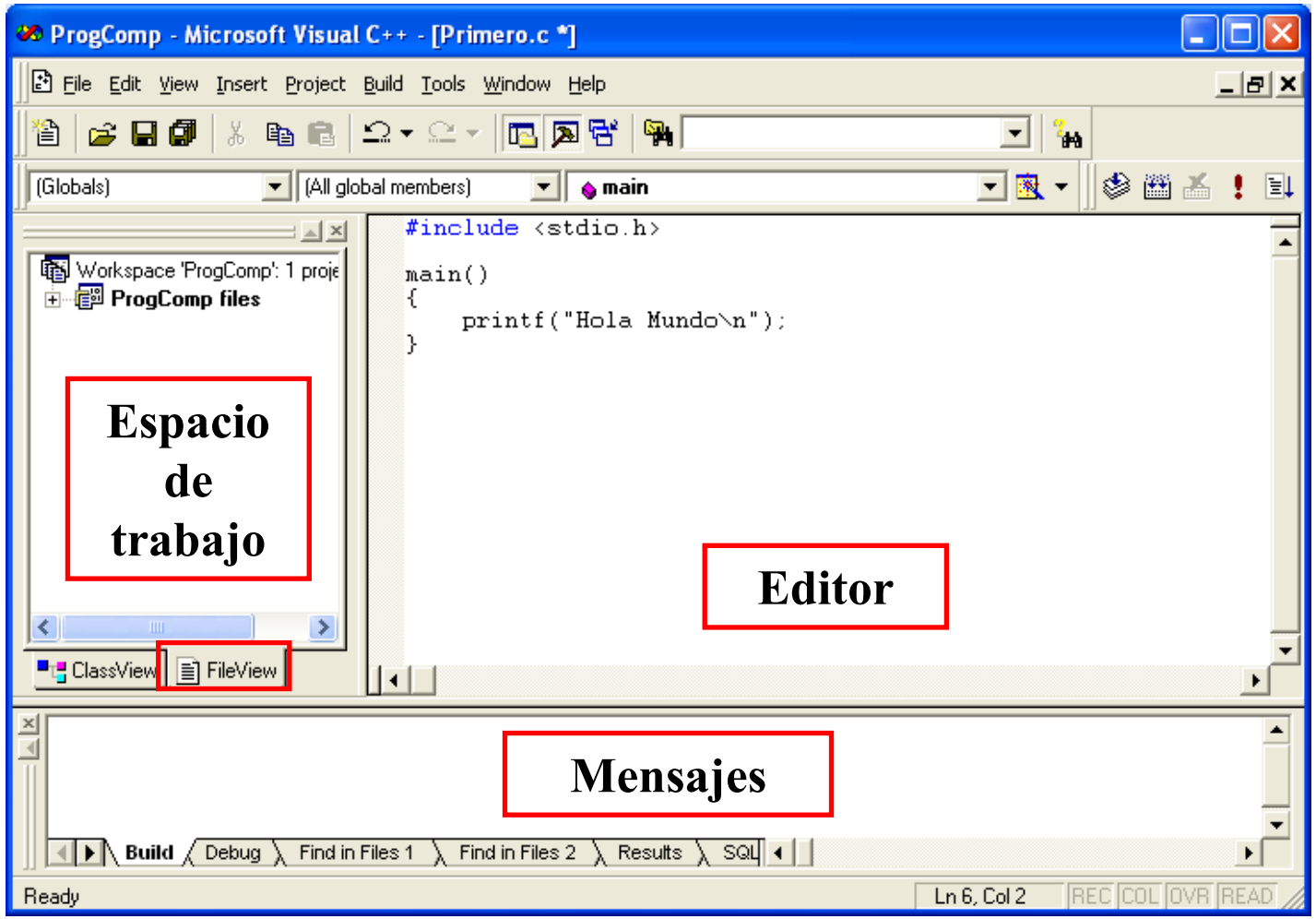
- Crear fichero fuente:
  - Pestaña: Files – Tipo: C++ Source File
  - Nombre Fichero (File name): Nombre_fich**.c** ← **Atención al sufijo!**





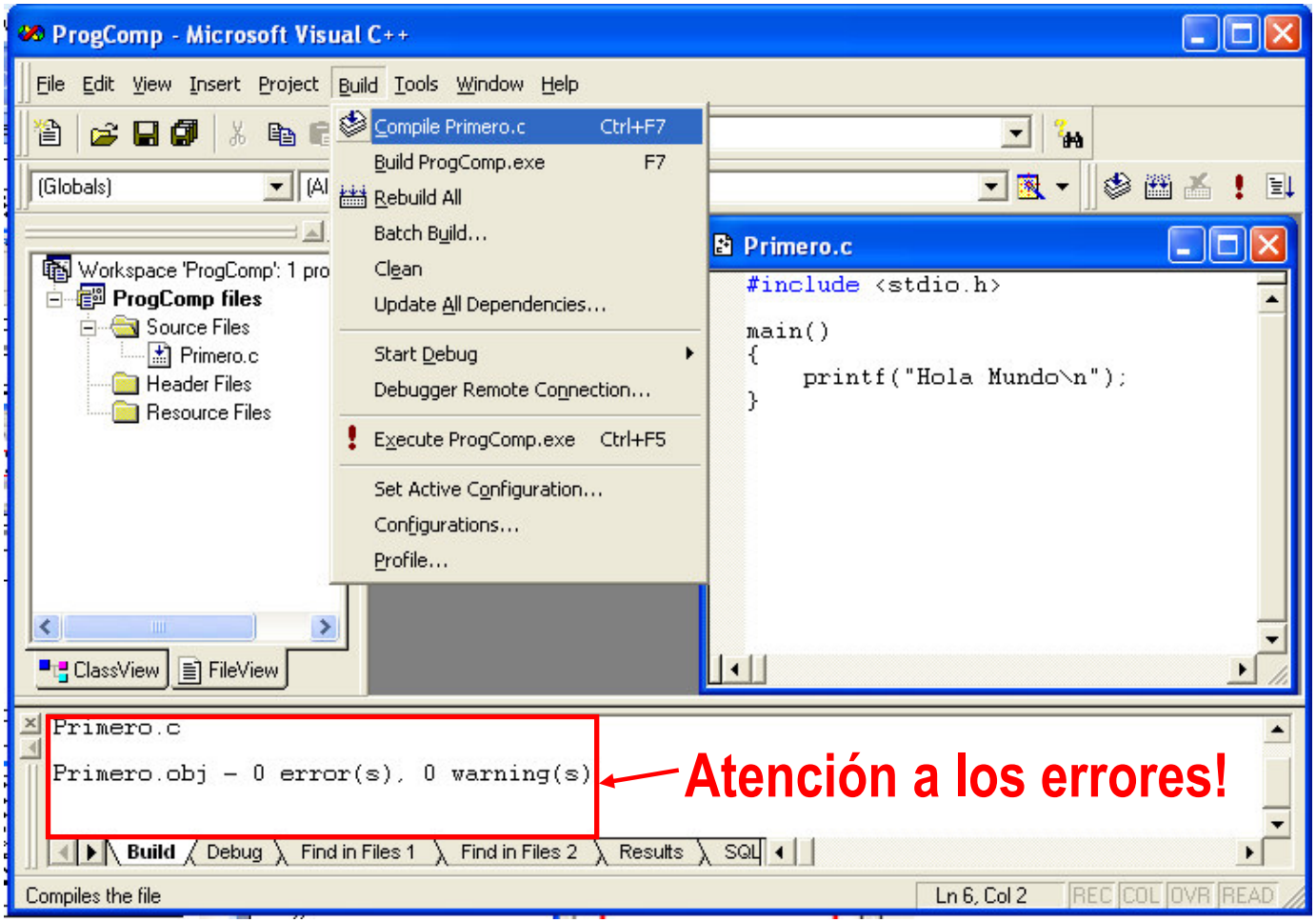
MS Visual Studio 6 - Edición

- Editar fichero fuente.



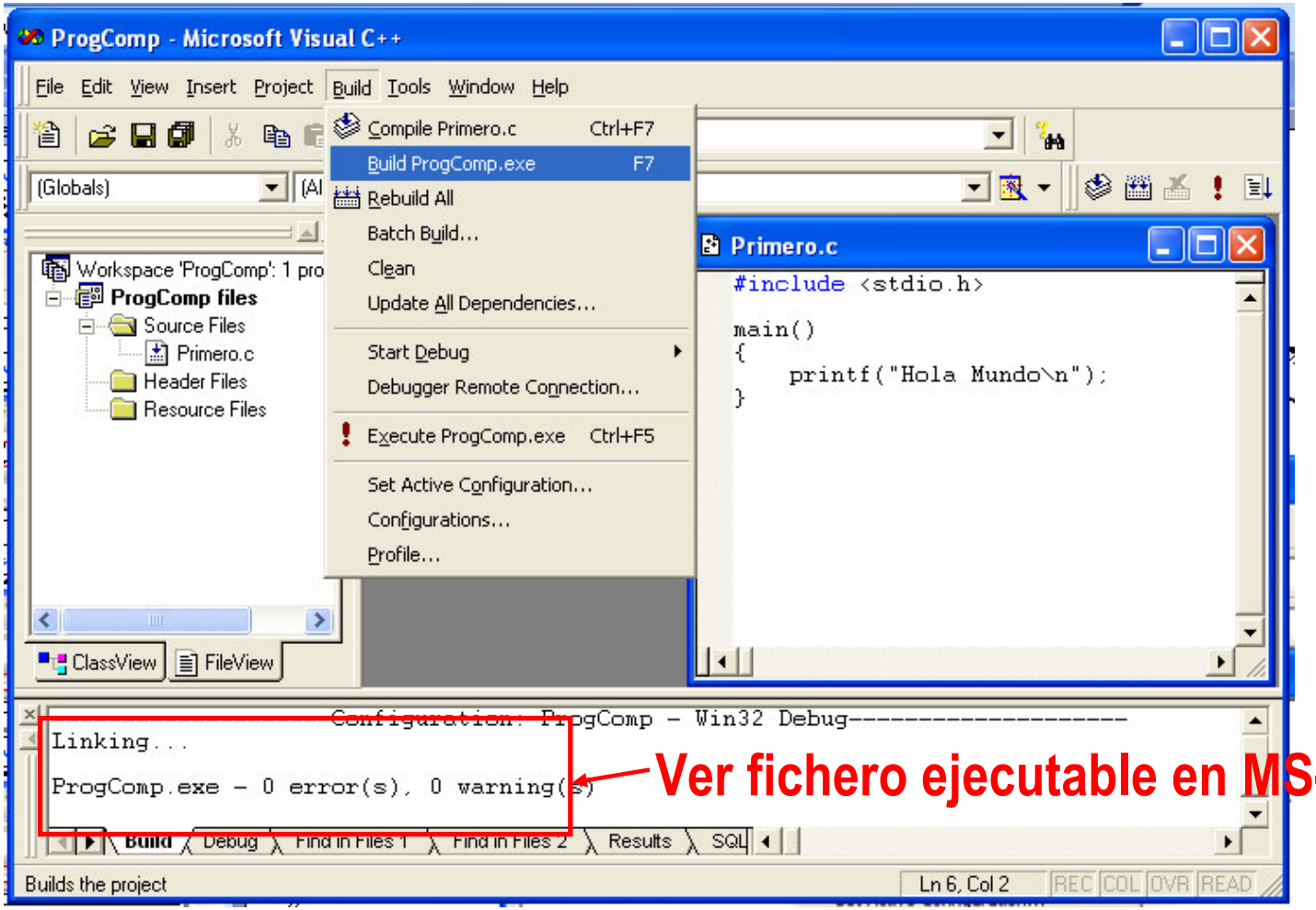
MS Visual Studio 6 - Compilación

- Compilar código fuente (Compile). Se crea fichero objeto.



MS Visual Studio 6 - Linkado

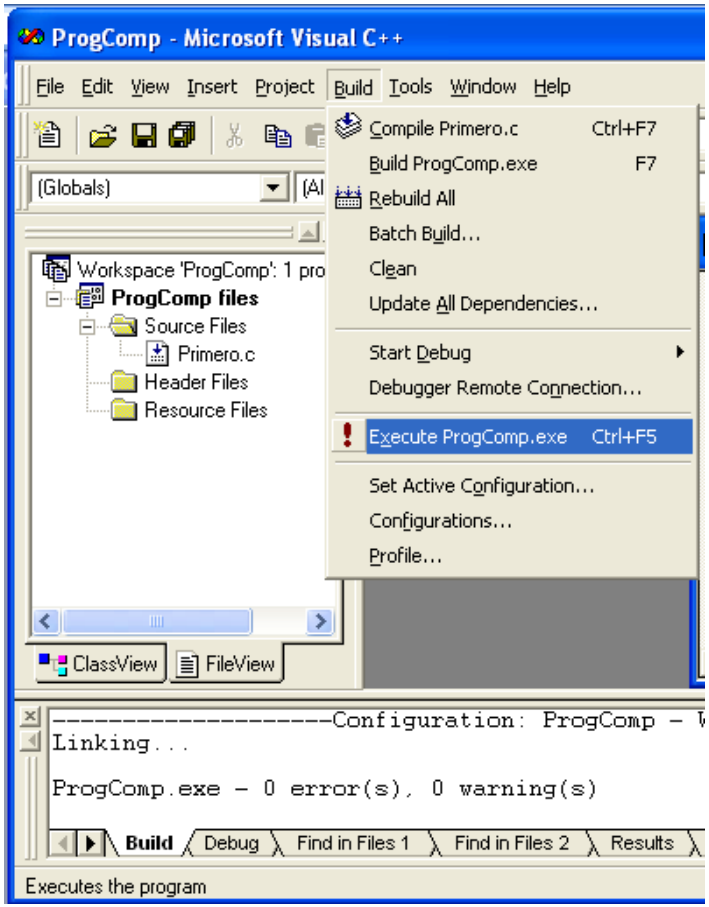
- Generar código máquina (Build). Se crea fichero ejecutable.



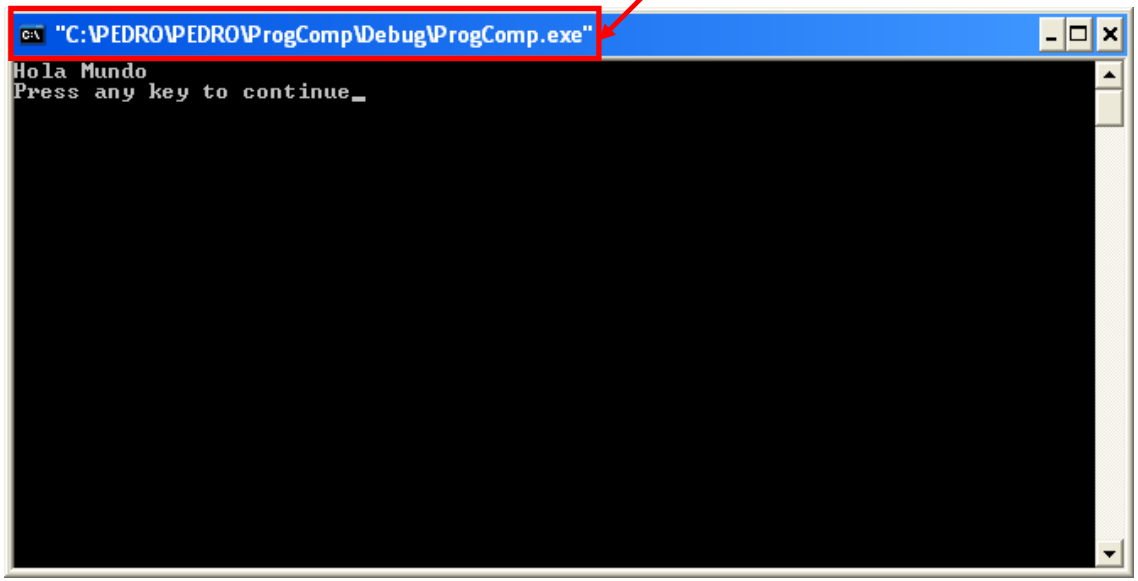
Ver fichero ejecutable en MS-DOS!

MS Visual Studio 6 - Ejecución

- Ejecutar código ejecutable (Execute). El fichero ejecutable tiene el nombre del proyecto.

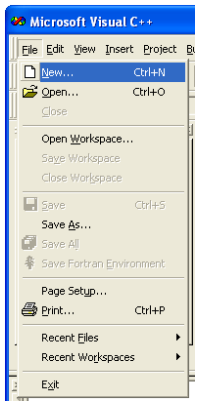


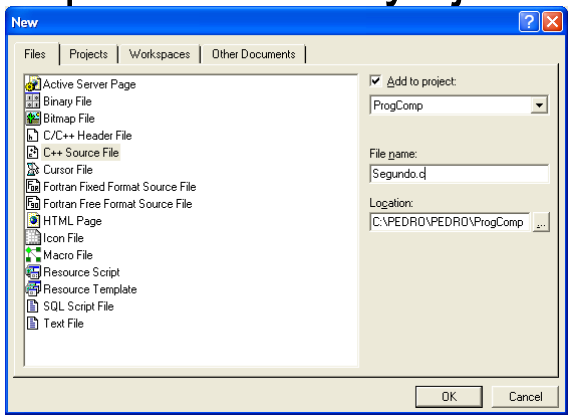
Ruta del ejecutable en MS-DOS

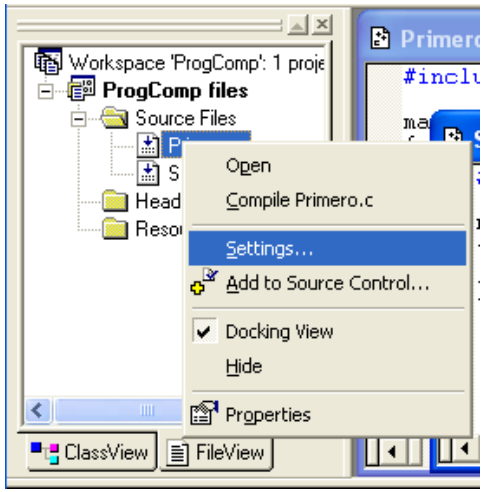


MS Visual Studio 6 – Añadir ficheros fuente

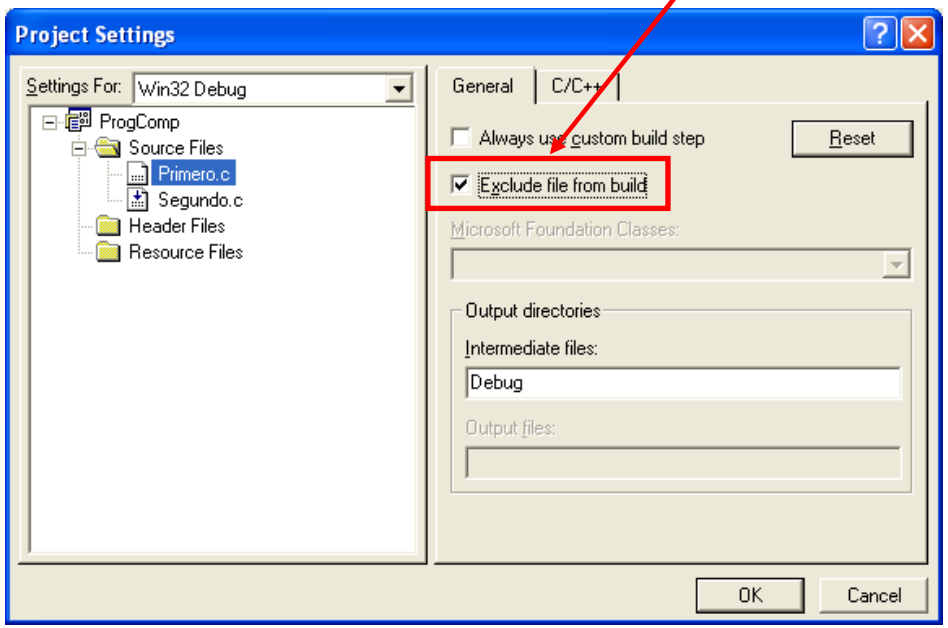
- Repetir proceso de creación de un fichero fuente.
- Sólo se puede Linkar y ejecutar un fichero fuente.







Opción para excluir del proceso de compilación

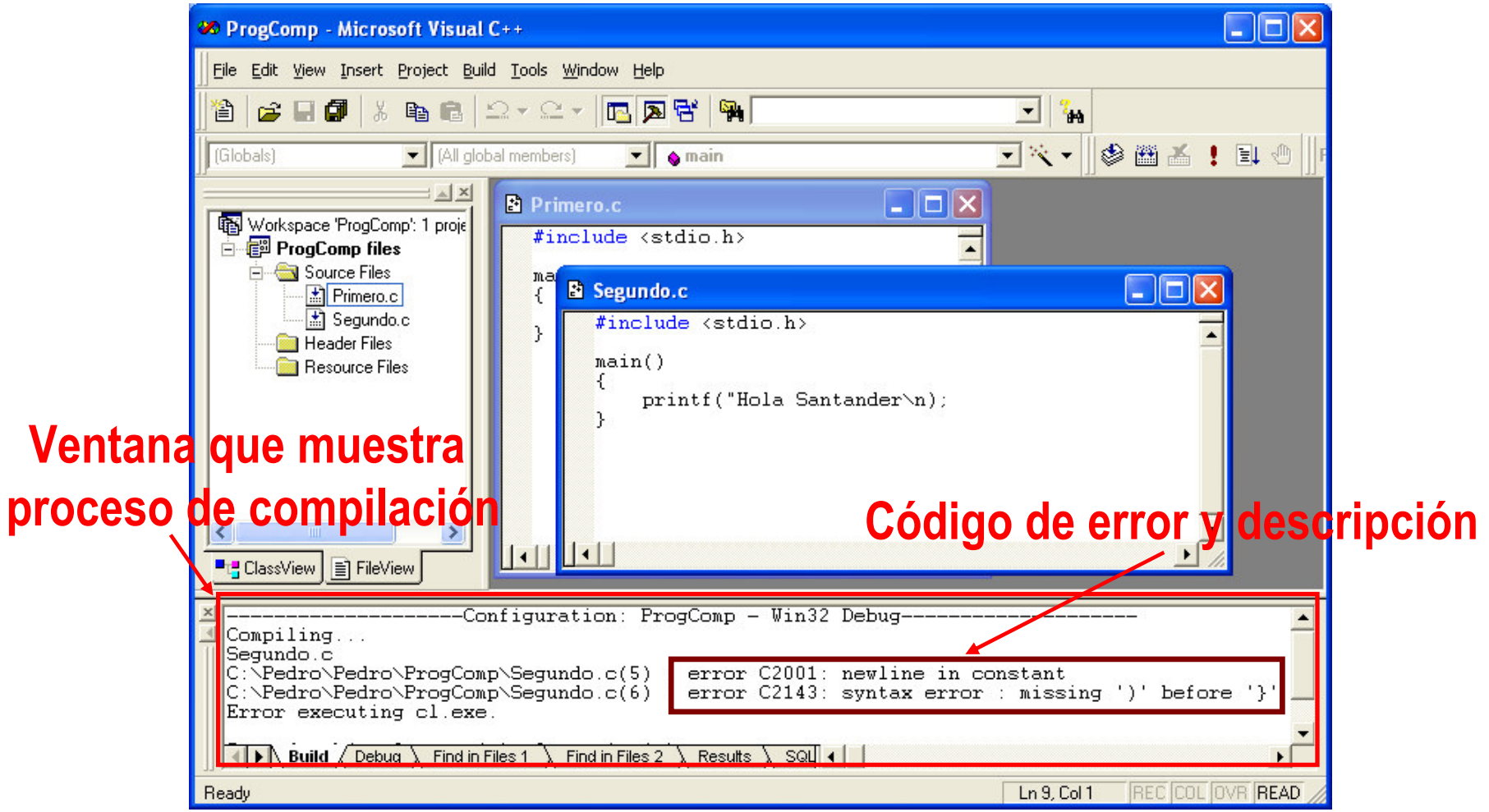


Depuración de programas

- Al proceso de detección y eliminación de errores de programación se le llama ***depuración*** (**debugging**).
- Los errores de programación pertenecen a tres categorías:
  - **errores de compilación**: son errores que impiden generación del programa ejecutable, debido a errores cometidos al escribir el código. Cuando se compila un programa con errores de compilación, el compilador los detecta y los muestra en la ventana de mensajes.

MS Visual Studio 6 – Errores de compilación

- Los errores tienen un código y una descripción.

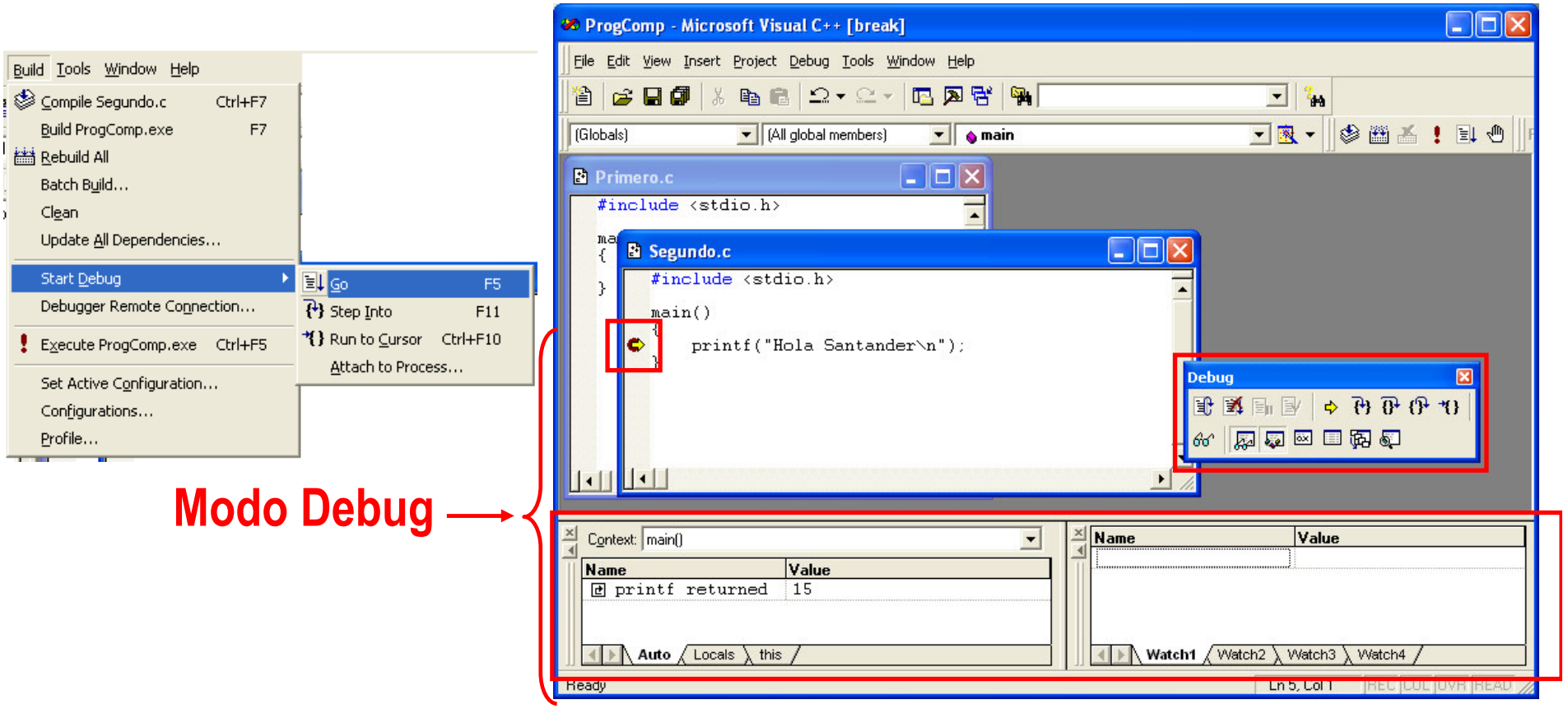


Depuración de programas

- Otros tipos de errores de programación:
  - **errores en tiempo de ejecución:** son errores que aparecen mientras se ejecuta su programa. Aparecen normalmente cuando un programa intenta una operación inválida. Un ejemplo de esto es la división por cero.
  - **errores lógicos:** son errores que impiden que un programa haga lo que estaba previsto. El código puede compilarse y ejecutarse sin errores, pero el resultado de una operación puede generar un resultado no esperado. Son los más difíciles de detectar y corregir.

MSVS 6 – Modo Debug

- El entorno ofrece la posibilidad de insertar puntos de parada, ejecución paso a paso, visualización/modificación de variables, etc.



Estructura de un programa C

- Un programa C consiste de una colección de **funciones** que interactúan entre sí.
- En todo programa debe haber una función especial llamada **main** que es la que se invoca primero.
- Una función C es una colección de operaciones en lenguaje C.

Primer ejemplo

```
/*****************************************\
* Programa: Hola                          *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>  /* Funciones Entrada/Salida  */

int main(void)
{
  printf("Hola PC! \n" );  /* Imprime mensaje */
  return 0;    /* Indica al SO que el programa ha
                  terminado sin error          */
}
```

Primer ejemplo

```
/*************************************************\
* Programa: Hola                                  *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                           *
* Revisión: 1.0 2/02/2008                         *
\*************************************************/

#include <stdio.h>   /* Funciones Entrada/Salida   */

int main(void)
{
  printf("Hola PC! \n" );  /* Imprime mensaje */
  return 0;        /* Indica al SO que el programa ha
                      terminado sin error          */
}
```

Comentarios

Primer ejemplo

```
/*****************************************\
* Programa: Hola                          *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>  /* Funciones Entrada/Salida  */

int main(void)
{
  printf("Hola PC! \n" );  /* Imprime mensaje */
  return 0;     /* Indica al SO que el programa ha
                   terminado sin error          */
}
```

Directiva del preprocesador para incluir contenido de un fichero

Primer ejemplo

```
/*****************************************\
* Programa: Hola                          *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>   /* Funciones Entrada/Salida   */

int main(void)
{
  printf("Hola PC! \n" );   /* Imprime mensaje */
  return 0;     /* Indica al SO que el programa ha
                   terminado sin error          */
}
```

Función principal obligatoria.
Tipo entero, sin parámetros

Primer ejemplo

```
/*****************************************\
* Programa: Hola                          *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>  /* Funciones Entrada/Salida  */

int main(void)
{
   printf("Hola PC! \n" );  /* Imprime mensaje */
   return 0;     /* Indica al SO que el programa ha
                    terminado sin error          */
}
```

El contenido de las funciones deben encerrarse entre llaves

Primer ejemplo

```
/*****************************************\
* Programa: Hola                          *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>  /* Funciones Entrada/Salida  */

int main(void)
{
  printf("Hola PC! \n" );  /* Imprime mensaje */
  return 0;     /* Indica al SO que el programa ha
                   terminado sin error          */
}
```

Instrucción para imprimir mensaje

Primer ejemplo

```
/*****************************************\
* Programa: Hola.c                        *
* Descripción: Escribe un mensaje en la pantalla  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>  /* Funciones Entrada/Salida  */

int main(void)
{
  printf("Hola PC! \n" );  /* Imprime mensaje */
  return 0;     /* Indica al SO que el programa ha
                   terminado sin error          */
}
```

Instrucción para salir de una función de forma normal

Primer ejemplo – Resumen (I)

- **Comentarios:** texto encerrado entre /* y */
  - Son ignorados por el compilador
  - Deben usarse para describir y documentar el código
- **#include <stdio.h>**
  - instruye al compilador que reemplace esa línea con el fichero stdio.h (contiene información respecto a las funciones E/S)
- **int main(void)**
  - Función principal en todo programa. Los paréntesis se usan para indicar una función.
  - int significa que la función devuelve un valor entero.
  - **void** indica que no requiere parámetros.
  - Las llaves { } indican un bloque.

Primer ejemplo – Resumen (II)

- **printf(“Hola PC! \n”);**
  - Función para imprimir con formato. Imprime la cadena de caracteres encerrada entre comillas (“ “).
  - \n es un carácter (secuencia de escape) que indica nueva línea.
  - Toda instrucción termina con un **punto y coma ';'**
- **return 0;**
  - Instrucción para salir de una función.
  - El valor 0 indica que el programa termina normalmente (convención). Valor devuelto al sistema operativo.

Segundo ejemplo

```
/*****************************************\
* Programa: promedio.c                    *
* Descripción: Lee numeros y calcula su promedio  *
* Autor: Pedro Corcuera                   *
* Revisión: 1.0 2/02/2008                 *
\*****************************************/

#include <stdio.h>

int main(void)
{
  int num_leidos = 0;       /* variable contador */
  float num, suma = 0.0; /* variables para calculos */
  printf("Escribir tantos numeros como quiera\n");
  printf("Finalizar con una marca de EOF (Ctrl-d)\n");
```

Segundo ejemplo

```
  while (scanf("%f", &num) == 1) /* instr. mientras */
  {
    num_leidos = num_leidos + 1; /* incrementa en 1 */
    suma = suma + num;           /* acumula numeros */
  }

  printf("El promedio de %d numeros leidos es %f \n",
         num_leidos, suma/num_leidos);

  return 0;
}
```

Segundo ejemplo – Detalle

- **int num_leidos = 0;**
  float num, suma = 0.0;

Tipos de datos
Nombres de variables
e inicialización

  - Es necesario declarar toda variable antes de ser usada, indicando su tipo de dato (int para enteros, float para reales).
  - Se puede inicializar una variable durante su declaración.
- **scanf("%f", &num)**
  - Función para leer valores de variables desde el teclado.
  - Parámetros: una cadena de control que especifica el formato y las direcciones de las variables a leer, obtenidas con el operador &.
  - Retorna: número de asignaciones realizadas, -1 si hay un final de lectura o 0 si hay un error de formato.

Segundo ejemplo – Detalle

- **while (scanf("%f",&num) == 1) { . . . . }**
  - == es un operador de comparación. Devuelve 1 si es verdadero 0 si es falso
  - while (expr_lógica) { } es una instrucción de repetición del código entre llaves mientras que el valor de la expresión lógica entre paréntesis sea verdadero (1).
- **num_leidos = num_leidos + 1;**

  suma = suma + num;
  - Expresiones de asignación. Usan los operadores = y +.

Segundo ejemplo – Detalle

- **printf ("El promedio de %d numeros leidos es %f \n", num_leidos, suma/num_leidos);**
  - La *cadena de control* de printf puede contener especificadores de conversión de formato que se corresponden con los parámetros que siguen a la cadena de control.
  - %d sirve para imprimir enteros (int). Se corresponde con la variable de tipo int num_leidos.
  - %f sirve para imprimir reales (float). Se corresponde con el resultado de la expresión suma/num_leidos.

Buenas prácticas de programación

- Incluir ***comentarios*** de forma apropiada. Toda función debe estar precedida de un comentario describiendo el propósito de la función
- ***Indentar*** (sangría) apropiadamente el código que compone una función. Contribuye a la legibilidad del código.
- Seleccionar nombres de variables con significado para contribuir a la legibilidad del código y autodocumentación.
- Utilizar letras minúsculas para los nombres de los identificadores (variables, funciones, etc.).
- Separar la parte declarativa de la parte de las instrucciones de una función con un espacio en blanco.
- Colocar un espacio después de cada coma (,) e instrucción y antes y después de un operador binario.

Errores comunes de programación

- Olvidarse de terminar una sentencia con ; (punto y coma).
- Usar letras mayúsculas en lugar de minúsculas.
- Colocar declaraciones de variables entre instrucciones.
- Olvidar uno ó dos de las comillas de la cadena de control en un printf o scanf.
- Olvidarse el carácter % en la especificación de formato en la cadena de control en un printf o scanf.
- Colocar un \n fuera de la cadena de control de un printf o scanf.
- Colocar espacios en blanco en operadores de dos símbolos.

Tipos de datos

Índice

- Conjunto de caracteres.
- Identificadores.
- Paabas resevadas.
- Variables.
- Tipos de datos.
- Definición de variables.
- Constantes.
- Expresiones.
- Operadores.

Conjunto de caracteres (I)

- La unidad básica de un programa C es un carácter.
- Los caracteres válidos son:
  - letras minúsculas a-z (alfabeto inglés, sin ñ!)
  - letras mayúsculas A-Z
  - los dígitos 0-9
  - espacios en blanco: blancos (barra espaciadora), cambios de línea (↵), tabuladores (→).
  - Símbolos especiales:

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

Conjunto de caracteres (II)

- Los caracteres básicos se combinan en ***tokens***, para formar palabras.
- Los tokens forman ***instrucciones***, que a su vez forman ***funciones***. Las funciones se combinan en uno o más ficheros fuentes para formar un ***programa***.
- Los espacios en blanco son importantes para separar símbolos, así como para aumentar la legibilidad y para distinguir entre tokens.

Secuencias de escape

- Son combinaciones de caracteres que representan espacios en blanco y caracteres no gráficos. Se usan dentro de cadenas de caracteres y constantes carácter.
- Consisten de una barra invertida (\) seguida de una letra o una combinación de dígitos.

Secuencia escape	Nombre
\n	Nueva línea
\t	Tabulador horizontal
\v	Tabulador vertical
\b	Retroceso
\r	Retorno de carro
\f	Salto de página
\a	Campana

Sec. escape	Nombre
\'	Comilla simple
\"	Comilla doble
\?	Pregunta
\\	Barra invertida
\ddd	Caracter ASCII not. octal
\xdd	Caracter ASCII en not. hexad.

Nota: ddd representa dígitos en la notación específica

Identificadores (I)

- Un **identificador** es una secuencia de caracteres (token) que nombra un objeto, una función u otros elementos de los programas C.
- Elementos que lo forman:

  Letras: a-z, A-Z

  Dígitos: 0-9

  Guión bajo: _
- Sintaxis: letra | _ [letra | digito | _]
- Longitud máxima de un identificador: Depende del compilador (usualmente 31 caract.).
- C distingue entre mayúsculas y minúsculas: Coef_Temp ≠ coef_temp

Identificadores (II)

- Ejemplos de identificadores válidos:

  numero fila error x14 codigosalida
  grados_celsius indice_array _9_ Double
- Ejemplos de identificadores no válidos:

  hola! default A-B 1lugar
- **Norma de estilo**: Usar identificadores que sean descriptivos (ayuda a la legibilidad, autodocumentación, depuración y modificación del código).

Palabras reservadas

- Son identificadores (32) predefinidos que tienen un significado especial para el compilador C y **no** se pueden usar para otro fin.
- A veces se usan en grupos

C99

```
_Bool    true  false
_Imaginary _Complex
```

```
auto      double    int       struct
break     else      long      switch
case      enum      register  typedef
char      extern    return    union
const     float     short     unsigned
continue  for       signed    void
default   goto      sizeof    volatile
do        if        static    while
```

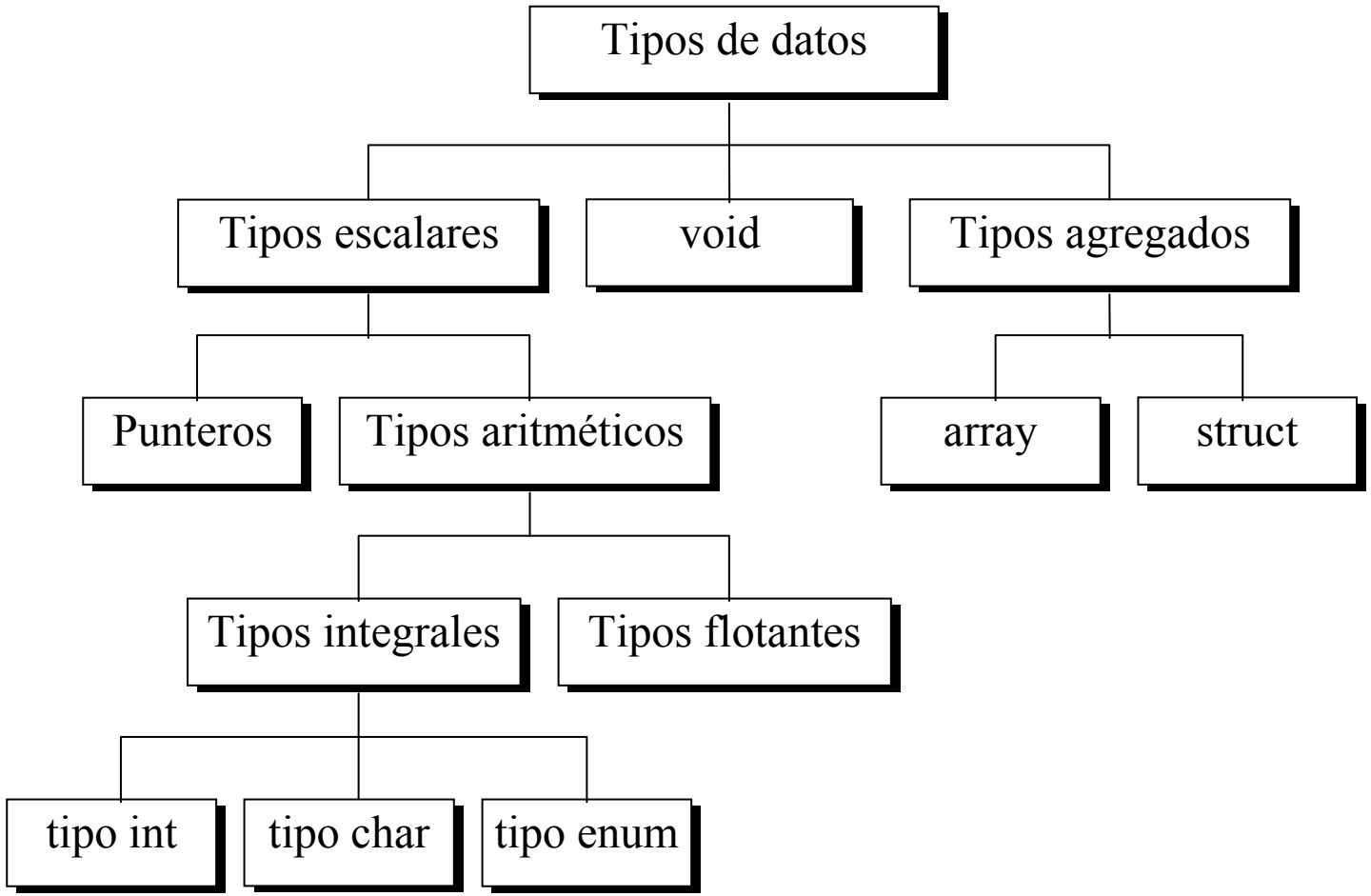
Variables

- Todos los datos que maneja un programa se almacenan en variables.
- Los nombres de variables corresponden a direcciones de memoria del ordenador.
- El nombre de una variable debe ser un identificador válido que no coincida con una palabra reservada.
- Cada variable tiene un nombre, un tipo de dato y un valor.
- Antes de usar una variable, hay que indicarle al compilador qué tipo de dato va a contener para que asigne la memoria necesaria y la forma de codificarlo en binario. Este proceso se conoce como **definición** de la variable, que debe realizarse al principio de la función, antes de las instrucciones.

Tipos de datos

- Un tipo de dato es una representación aplicada a una cadena de bits.
- Hay tipos diferentes de enteros y tipos de reales (puntos flotantes).
- Los tipos, enteros y flotantes, son llamados tipos **aritméticos**. Junto con los punteros y tipos enumerados forman los tipos **escalares** porque todos los valores pueden representarse en una escala lineal.
- Adicionalmente hay tipos agregados, que son combinaciones de uno o más tipos de datos escalares (arrays, las estructuras y uniones).
- El tipo void no es ni escalar ni agregado. Se aplica sólo a funciones y punteros.

Tipos de datos



Tipos de datos aritméticos

- char, int, float, double, enum. **Tipos básicos**
- Especificadores adicionales: short, long, signed, unsigned.

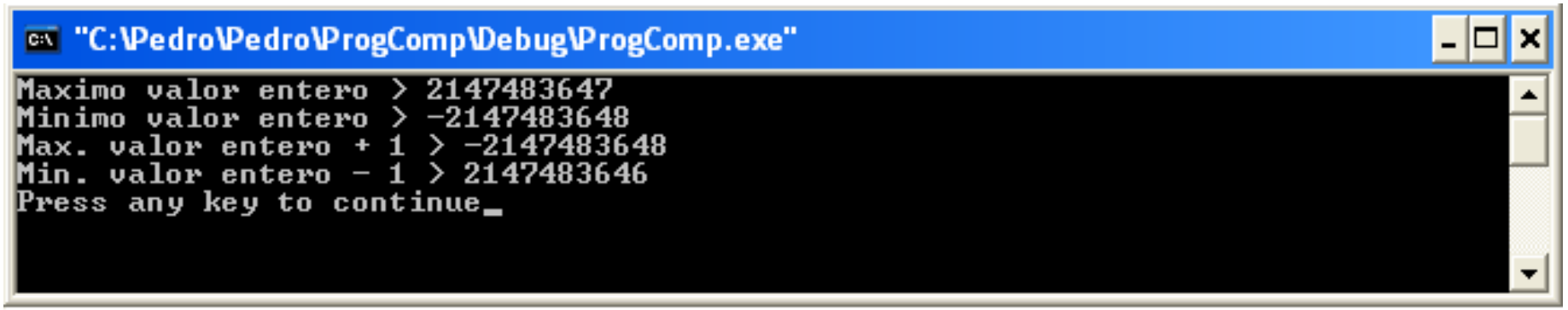
Tipo	Tamaño (bytes)	Rango de valores
char, signed char	1	-128 a 127
unsigned char	1	0 a 255
short int	2	-2^{15} a $2^{15}-1$
unsigned short	2	0 a $2^{16}-1$
int, long	4	-2^{31} a $2^{31}-1$
unsigned int, unsigned long	4	0 a $2^{32}-1$
float	4	$1.17\text{x}10^{-38}$ a $3.4\text{x}10^{+38}$ (8dp)
double, long double	8	$2.2\text{x}10^{-308}$ a $1.7\text{x}10^{+308}$ (16dp)

Ejemplo - Rango de enteros

```
/*********************************************\
* Programa: rango_enteros.c                   *
* Descripción: Imprime el maximo y minimo de  *
* valores enteros. Las constantes simbolicas estan*
* definidas en limits.h                       *
* Autor: Pedro Corcuera                       *
* Revisión: 1.0 2/02/2008                     *
\*********************************************/

#include <stdio.h>
#include <limits.h>

main()
{
 int respuesta;

 printf("Maximo valor entero > %d\n", INT_MAX );
 printf("Minimo valor entero > %d\n", INT_MIN );
```

Rango de enteros

```
 /* que pasa si se anade/resta 1? */
 respuesta = INT_MAX + 1;
 printf("Max. valor entero + 1 > %d\n",respuesta);
 respuesta = INT_MAX - 1;
 printf("Min. valor entero - 1 > %d\n",respuesta);

 return 0;
}
```



Definición de variables (I)

- Toda variable debe declararse antes de su uso. Sirve para especificar el tipo de dato de la variable. Debe colocarse al inición de la función.
- Sintaxis general de una definición:

  tipo_dato nombre_variable [= const/expresion_inicialización];
- Ejm:

```
int bajo, alto, paso;
char c, linea[100];
```

- Una variable se puede inicializar durante su definición. Se usa el operador de asignación y una expresión o constante. Ejm:

```
int bajo = -10, alto = 10, paso = 1;
char esc = '\\';
int pp = 0;
float limite = MAXLINE + 1;
float eps = 1.0e-5;
```

Definición de variables (II)

- Las variables (*automáticas*) no inicializadas tienen cualquier valor.
- Una variable (*automática*) se inicializa cada vez que se llama a la función o bloque donde se declara.
- Las variables externas y estáticas se inicializan a cero por defecto.
- Se puede agregar el calificador **const** a la declaración de una variable para especificar que su valor no será modificado. Ejm:

```
const double e = 2.718281828459O5;
const char msg[ ] = "cuidado: ” ;
```

Tipos de variables

- Los tipos elementales de variables en C son:
  - Enteros (`int`).
  - Reales (`float`, `double`).
  - Caracteres (`char`).
  - Punteros (`*`).

NO existe un tipo booleano (en su lugar se usa `int` o `char`).

Modificadores de tipos

- Ciertos tipos básicos admiten diversos modificadores:
  - `unsigned` :Sólo valores positivos (sin signo).
  - `signed` : Valores positivos y negativos (por omisión).
  - `long` : Formato largo (para enteros únicamente).

Ejemplos:

```
unsigned int
signed char
long int (usualmente representado como long)
unsigned long int
```

Declaración de variables

- Declaración simple:
  - `char c;`
  - `unsigned int i;`
- Declaración múltiple:
  - `char c,d;`
  - `unsigned int i,j,k;`
- Declaración y asignación:
  - `char c='A';`
  - `unsigned int i=133,j=1229;`

Constantes

- Las constantes son cualquier número que se escribe en el código, carácter o cadenas de caracteres.
- C tiene cuatro tipo básicos de constantes:
  - Constantes enteras.
  - Constantes en coma flotante.
  - Constantes de carácter.
  - Constantes de cadena de caracteres.
- Hay también constantes enumeradas que son un tipo de constantes simbólicas.

Constantes enteras

- Una constante entera es un número con un valor entero, consistente en una secuencia de dígitos.
- Se pueden expresar en tres sistemas numéricos:
  - decimal: digitos[0-9]
  - octal: 0digitoso[0-7]
  - hexadecimal: 0{x|X}digitosh[0-F]
- Ejemplos: `32767  0743  0x7FFF`
- Se puede forzar cualquier constante entera a que sea de tipo long agregando una l o L al final de la constante, ej. `123456789L`.
- Una constante sin signo lleva como sufijo u o U, ej. `0xffffffUL`.

Constantes en coma flotante

- Una constante en coma flotante es un número en base 10 que contiene un punto decimal o un exponente (o ambos). Representan los números reales.
- Sintaxis: [digitos][.digitos][e|E[-|+]digitos]
- Ejemplos: `327.67  2E-6  1.212e+5`
- Se puede forzar que un número sea de punto flotante colocando un sufijo f o F, ej. `3E5L`.
- La precisión varía según el compilador, pero lo habitual es seis cifras significativas.
- Las constantes en coma flotante son aproximaciones por lo que no deben usarse para contadores, índices, etc. Donde se requiere valores exactos (enteros).

Constantes de carácter

- Una constante de carácter es un solo carácter, encerrado con comillas simples.
- Sintaxis: `'char'`
- Ejemplos: `'A'  'X'  '3'  ' '`
- Se utiliza el conjunto de caracteres del código ASCII. Una secuencia de escape se considera como un caracter simple.
- El valor de una constante caracter es el valor numérico del carácter en el código ASCII. La constante caracter '\0' representa el caracter con valor cero (caracter nulo). Ej.: el valor de 'A' es 65, de 'a' es 97, de '0' es 48.

Constantes de cadena de caracteres

- Una constante de cadenas de caracteres consta de cualquier número de caracteres consecutivos (o ninguno) encerrado entre comillas dobles.
- Sintaxis: `"caracteres"`
- Ejemplos: `"azul" "19.95e5"`
  `"L1\tL2\\L3\"L4\n"`
- Los caracteres se almacenan en posiciones contiguas de memoria. El compilador añade un caracter nulo ('\0') para marcar el final de la cadena.

Constantes enumeradas

- Consisten de una lista de valores enteros constantes etiquetados.
- Sintaxis:

```
enum [identificador]{lista de enumerados}
```

- Ejemplos:

```
enum colores { rojo, azul, verde, amarillo }
enum {manzana=0, naranja=10, limon= -5, uva}
```

- El primer nombre la lista de enumerados tiene valor 0, el siguiente 1 y así sucesivamente, a menos que se asignen valores explícitos.
- Si no se especifican todos los valores, los valores sin especificar continuan con la progresión a partir del último valor especificado.

Constantes simbólicas

- Una constante simbólica es un nombre que sustituye una secuencia de caracteres.
- Es gestionado por el *preprocesador* quien reemplaza la constante simbólica por su correspondiente secuencia de caracteres.
- Se definen al comienzo del programa (después de los includes) y el nombre generalmente se escribe en letras mayúsculas.
- Sintaxis:

```
#define nombre texto
```

Ejemplos de constantes simbólicas

```
#define INTERES 0.16
#define PI 3.141593
#define TRUE 1
#define FALSE 0
#define AMIGO “Obama”
....
area = PI * radio * radio;
printf(“El interes es %f”, INTERES);
```

Expresiones

- Es la combinación de **operandos** y **operadores** que indican cómo realizar un cálculo y produce un valor simple.
- Un operando es un valor constante o variable.
- Cada operando de una expresión es también una expresión, ya que representa una valor simple.
- El resultado de una expresión depende de la *precedencia* y de la *asociatividad* de los operadores.
- C ofrece un amplio repertorio de operadores de asignación.

Mezcla de tipos en expresiones

- Se pueden mezclar tipos de datos aritméticos en una expresión.
- C realiza la siguientes *conversiones implícitas* (silenciosas o automáticas) :
  - En una asignación, el valor de la derecha se convierte al tipo de dato de la variable a la izquierda.
  - Si char o short int aparece en una expresión, se convierte a un entero.
  - En expresiones aritméticas los objetos se convierten según las reglas del operador. Ejm: int j=2.6; (j = 2)

Conversiones explícitas

- Una expresión se divide en subexpresiones, donde cada subexpresión consta de un operador y uno o más operandos.

  Ejm: –3 / 4 + 2.5 Operadores: – / +

  Operandos: 3 4 2.5

- Si los tipos de datos de los operandos difiere, se convierte uno de los operandos para compatilizarlo con el otro, siguiendo una jerarquía.
- Una conversión explícita (*cast*) consiste en forzar la conversión de tipo de una constante o expresión anteponiendo el tipo de dato deseado.

  Ejm: x = (float) 2;

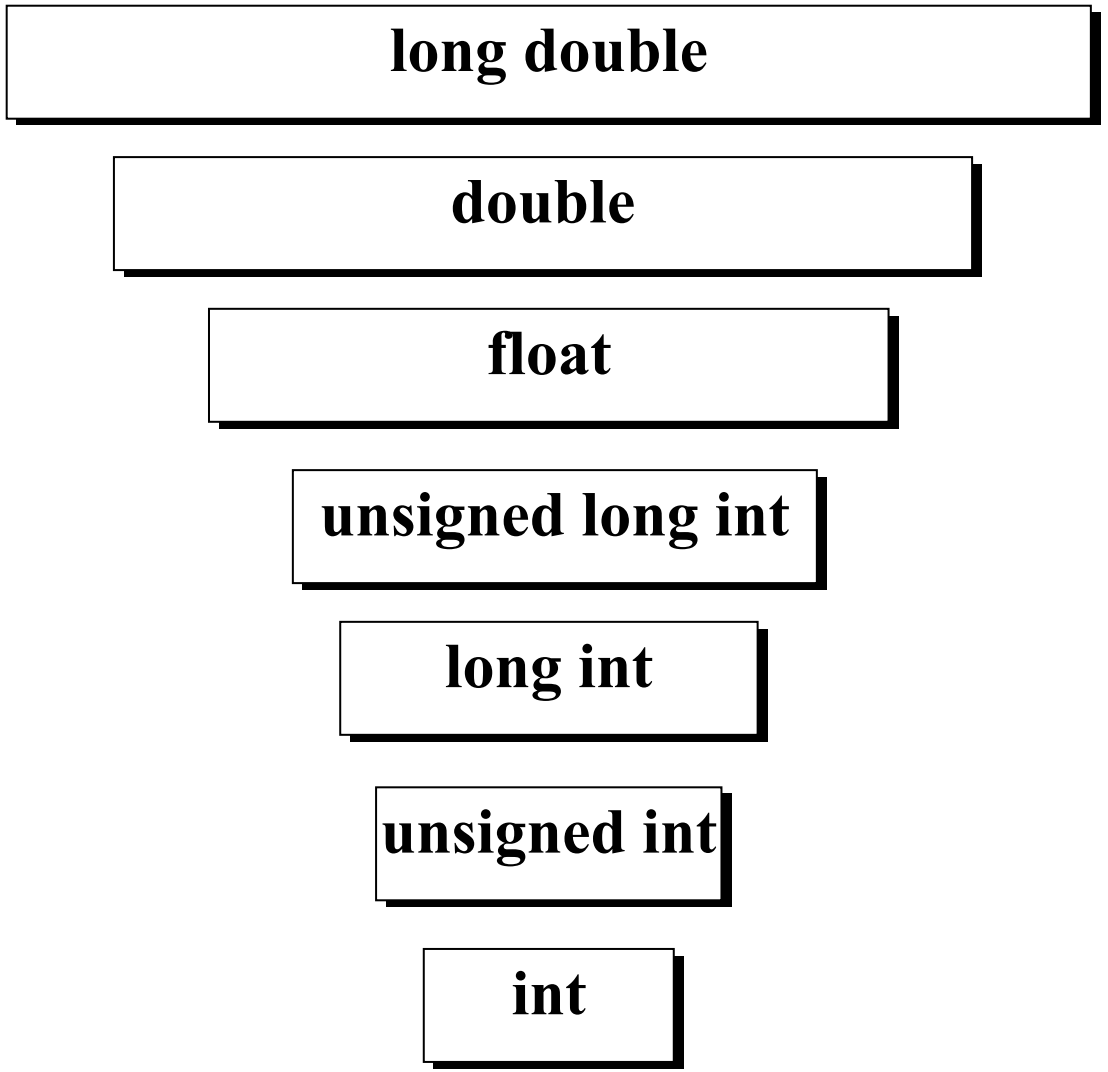
  f = (float) j / k;

Casting

- *Casting*: mecanismo usado para cambiar de tipo expresiones y variables:

```
int a;
float b;
char c;

b=65.0;
a=(int)b;    /* a vale 65 */
c=(char)a;   /* c vale 65 (Código ASCII
                de 'A') */
```

Jerarquía de conversiones



Operadores

- C ofrece un amplio repertorio de operadores.
- El resultado de una expresión depende de la *precedencia* y de la *asociatividad* de los operadores.
- Lista de operadores:

Operadores aritméticos binarios
Operadores unarios + y -
Operadores de asignación aritméticos
Operadores de incremento, decremento
Operador coma
Operadores relacionales
Operadores lógicos
Operadores para manipulación de bits
Operadores para asignación de bits
Operador condicional (ternario)
Paréntesis
Operador cast
Operador sizeof
Operadores para manejo de la memoria

Asociatividad y precedencia de operadores

Clase de Oper.	Operadores	Asociatividad	Precedencia
primario	( ) [ ] -> .	Izq. a der.	Más alta
unario	op.cast sizeof & * – + ~ ++ -- !	Der. a izq.	
multiplicativo	* / %	Izq. a der.	
aditivo	+ –	Izq. a der.	
desplazam. bits	<< >>	Izq. a der.	
relacional	< <= > >=	Izq. a der.	
comparación	== !=	Izq. a der.	
manejo de bits	& ^ |	Izq. a der.	
lógicos	&& ||	Izq. a der.	
condicional	? :	Der. a izq.	
asignación	= += –= *= /= %= >>= <<= &= ^=	Der. a izq.	
coma	,	Izq. a der.	Más baja

Operadores aritméticos binarios

- Es la combinación de **operandos** y **operadores** que indican cómo realizar un cálculo y produce un valor simple.

Operador	Símbolo	Sintaxis	Operación
multiplicación	*	x * y	x veces y
división	/	x / y	x dividido por y
resto	%	x % y	resto de x / y
suma	+	x + y	x más y
resta	–	x – y	x menos y

Operaciones aritméticas

- División entera vs división real:
  - Depende de los operandos:

```
4 / 3      --> 1          entero
4.0 / 3    --> 1.333      real
4 / 3.0    --> 1.333      real
4.0 / 3.0  --> 1.333      real
```

Ejemplo: Operadores aritméticos binarios

- Dadas las siguientes declaraciones:

```
int m = 3, n = 4;
float x = 2.5, y = 1.0;
```

Expresión	Expresión equiv.	Resultado
m + n + x + y		
m + n * x + y		
x / y + m / n		
x – y * m + y / n		
x / 0		

Ejemplo: Operadores aritméticos binarios

- Dadas las siguientes declaraciones:

```
int m = 3, n = 4;
float x = 2.5, y = 1.0;
```

Expresión	Expresión equiv.	Resultado
m + n + x + y	(((m + n) + x) + y)	10.5
m + n * x + y	((m + (n * x) + y))	14.0
x / y + m / n	(x / y) + (m / n)	2.5
x – y * m + y / n	(x – (y * m)) + (y / n)	-0.25
x / 0	x / 0	1.#INF00

Ejemplo: Operadores aritméticos binarios

```
/*************************************************************************\
* Programa: imprime_aintervalos.c                                         *
* Descripción: Uso del operador % para realizar una acción en intervalos *
* regulares: lee una cadena de caracteres y la imprime insertando un      *
* cambio de línea cada 5 caracteres.                                      *
* Autor: Pedro Corcuera                                                   *
* Revisión: 1.0 2/02/2008                                                 *
\*************************************************************************/

#include <stdio.h>

int main( void )
{
  int c, j = 0;

  printf( "Ingresa cadena a ser partida ");
  while ((c = getchar()) != '\n')
  {
    if ( j % 5 == 0) /* j divisible por 5? */
      printf( "\n" );
    putchar( c );
    j++;
  }
  return 0;
}
```

Operadores unarios + y –

Operador	Símbolo	Sintaxis	Operación
menos unario	–	– x	negación de x
más unario	+	+ x	valor del operando

- Ejm: j = 3 – –x

Operadores de asignación aritméticos

- Tienen menor precedencia y la asociatividad es de derecha a izquierda

Operador	Símbolo	Sintaxis	Operación
asignación	=	x = y	coloca valor de y en x
adición – asig.	+=	x += y	coloca valor de x+y en x
resta – asig.	–=	x –= y	coloca valor de x-y en x
multipl – asig.	*=	x *= y	coloca valor de x*y en x
división – asig.	/=	x /= y	coloca valor de x/y en x
resto – asig.	%=	x %= y	coloca valor de x%y en x

Ejemplo: Operadores de asignación aritméticos

- Dadas las siguientes declaraciones:

```
int m = 3, n = 4;
float x = 2.5, y = 1.0;
```

Expresión	Expresión equiv.	Resultado
m += n + x – y		
m /= n * x + y		
n %= (int) y + m		
x += y –= m		

Ejemplo: Operadores de asignación aritméticos

- Dadas las siguientes declaraciones:

```
int m = 3, n = 4;
float x = 2.5, y = 1.0;
```

Expresión	Expresión equiv.	Resultado
m += n + x – y	m = (m + ((n + x) – y)	8
m /= n * x + y	m = (m / ((x * n) + y))	0
n %= (int) y + m	n = (n % (y + m))	0
x += y –= m	x = (x + (y = (y – m)))	0.5

Operadores de incremento, decremento

- Son operadores unarios que pueden ser prefijos o postfijos

Operador	Símbolo	Sintaxis	Operación
incr. postfijo	++	a++	obtiene valor de a, después incrementa a en 1
decr. postfijo	--	a--	obtiene valor de a, después decrementa a en 1
incr. prefijo	++	++a	incrementa a, después obtener valor de a en 1
decr. prefijo	--	--a	decrementa a, después obtener valor de a en 1

Operadores de incremento, decremento

Los operadores unarios (++) y (--) representan operaciones de incremento y decremento, respectivamente.

```
a++; /* similar a a=a+1 */
```

Ejemplos:

```
a=3; b=a++;     /* a=4, b=3 */
a=3; b=++a;     /* a=4, b=4 */
a=3; b=a--;     /* a=2, b=3 */
```

Ejemplo: Operadores de incremento, decremento

- Dadas las siguientes declaraciones:

```
int j = 0, m = 1, n= -1;
```

Expresión	Expresión equiv.	Resultado
m++ – --j		
m += ++j * 2		
m++ * m++		

Ejemplo: Operadores de incremento, decremento

- Dadas las siguientes declaraciones:

```
int j = 0, m = 1, n= -1;
```

Expresión	Expresión equiv.	Resultado
m++ – --j	(m++) – (--j)	2, m=2, j= – 1
m += ++j * 2	m = (m + ((++j) * 2)	3, m=3, j=1
m++ * m++	(m++) * (m++)	1, m=3

Ejemplo: Operadores de incremento, decremento

```
/*****************************************************************\
* Programa: operadores_incr_decr.c                                *
* Descripción: Uso de los operadores de incremento/decremento     *
*              sufijo/postfijo                                    *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/

#include <stdio.h>

int main( void )
{
  int j = 5, k = 5;

  printf( "j: %d\t k: %d\n", j++, k-- );
  printf( "j: %d\t k: %d\n", j, k );

  j = 5; k = 5;
  printf( "j: %d\t k: %d\n", ++j,--k );
  printf( "j: %d\t k: %d\n", j, k );

  return 0;
}
```

```
j: 5     k: 5
j: 6     k: 4

j: 6     k: 4
j: 6     k: 4
```

Operador coma

- Se utiliza con frecuencia en la sentencia for (repetición)

Operador	Símbolo	Sintaxis	Operación
coma	,	a, b	evalúa a, evalúa b, resultado es b

```
/* para insertar blancos cada valor de interval */
for (c=getchar(), j = 0; c != EOF; j++, c =getchar())
{
   putchar(c);
   if (j%interval == 0)
     putchar( '\n' );
}
```

Operadores relacionales

- Se utilizan para formar expresiones lógicas que representan condiciones que pueden ser verdaderas o falsas. La expresión resultante es de tipo entero, ya que *verdadero* se representa por 1 y *falso* por 0 en C.

Operador	Símbolo	Sintaxis	Operación
mayor que	>	a > b	1 si a es mayor que b, sino 0
menor que	<	a < b	1 si a es menor que b, sino 0
mayor o igual a	>=	a >= b	1 si a es mayor o igual a b, sino 0
menor o igual a	<=	a <= b	1 si a es menor o igual a b, sino 0
igual a	==	a == b	1 si a es igual a b, sino 0
no igual a	!=	a != b	1 si a no es igual a b, sino 0

Operadores relacionales

- en caso de comparar valores reales (float) no usar el operador de comparación exacta (==) debido a los errores de representación de los números flotantes.

  (1.0/3.0 + 1.0/3.0 + 1.0/3.0) == 1.0 → produce 0

Ejemplo: Operadores relacionales

- Dadas las siguientes declaraciones:

```
int j=0, m=1, n=-1;
float x = 2.5, y = 0.0;
```

Expresión	Expresión equiv.	Resultado
j > m		
m / n < x		
j <= m >= n		
j <= x == m		
-x + j == y > n > m		
x += (y >=n)		
++ j == m != y * 2		

Ejemplo: Operadores relacionales

- Dadas las siguientes declaraciones:

```
int j = 0, m = 1, n = -1;
float x = 2.5, y = 0.0;
```

Expresión	Expresión equiv.	Resultado
j > m	j > m	0
m / n < x	(m / n) < x	1
j <= m >= n	((j <= m) >= n)	1
j <= x == m	((j <= x) == m)	1
-x + j == y > n > m	((-x) + j) == (y > n) > m)	0
x += (y >=n)	x = (x + (y >=n))	3.5
++ j == m != y * 2	((++j) == m) != (y * 2)	1

Operadores lógicos

- Actuan sobre operandos que son a su vez expresiones lógicas.

Operador	Símbolo	Sintaxis	Operación
AND lógico	&&	a && b	1 si a y b son 1, sino 0
OR lógico	||	a || b	1 si a o b son 1, sino 0
negación lógica	!	!a	1 si a es cero, sino 0

- Tener en cuenta la evaluación en corto circuito de los operadores relacionales para evitar efectos secundarios. Así en: `if ((a < b) && (c == d++))` d sólo se incrementa en caso que a sea menor que b.

Ejemplo: Operadores lógicos

- Dadas las siguientes declaraciones:

```
int j = 0, m = 1, n = -1;
float x = 2.5, y = 0.0;
```

Expresión	Expresión equiv.	Resultado
j && m		
j < m && n < m		
m + n || !j		
x * 5 && 5 || m / n		
!x || !n || m + n		
(j || m) + (x || ++ n)		

Ejemplo: Operadores lógicos

- Dadas las siguientes declaraciones:

```
int j = 0, m = 1, n = -1;
float x = 2.5, y = 0.0;
```

Expresión	Expresión equiv.	Resultado
j && m	(j) && (m)	0
j < m && n < m	(j < m) && (n < m)	1
m + n || !j	(m + n) || (!j)	1
x * 5 && 5 || m / n	((x * 5) && 5) || (m / n)	1
!x || !n || m + n	((!x) || (!n)) || (m + n)	0
(j || m) + (x || ++ n)	(j || m) + (x || (++n))	2

Operadores para manipulación de bits

- Se aplican sobre operandos de tipo integral. El operando izquierdo se promociona a un entero.

Operador	Símbolo	Sintaxis	Operación
desplazam. der.	>>	x >> y	x desplaz. der. y bits
desplazam. izq.	<<	x << y	x desplaz. izq. y bits
AND nivel bits	&	x & y	x ANDbits y
OR inclusivo bits	|	x | y	x ORbits y
OR exclusivo	^	x ^ y	x exORbits y
complemento bits	~	~x	complemento bits de x

Operadores para manipulación de bits

- Cuando un entero positivo se desplaza a la izquierda o derecha los espacios vacantes se llenan con ceros.
- Cuando un entero negativo se desplaza a la izquierda o derecha los espacios vacantes se llenan con ceros (desplazamiento lógico) o unos (desplazamiento aritmético).
- Equivalencias:

  $x << y \equiv x * 2^y$

  $x >> y \equiv x / 2^y$

Ejemplo: Operadores para manipulación de bits

Expresión	Expresión equiv.	Resultado
5 << 1	00000000 00000101	
	00000000 00001010	10
255 >> 3	00000000 11111111	
	00000000 00011111	31
9430 0x24D6	00100100 11010110	
5722 0x165A	00010110 01011010	
0x24D6 & 0x165A	00000100 01010010	0x0452
0x24D6 | 0x165A	00110110 11011110	0x35DE
0x24D6 ^ 0x165A	00110010 10001100	0x328C
~0x24D6	11011011 00101001	0xDB29

Ejemplo: Operadores para manipulación de bits

```
/*****************************************************************\
* Programa: cuenta_bits.c                                         *
* Descripción: Cuenta el numero de bits a 1 en un entero          *
* Autor: Pedro Corcuera                                           *
* Revisión: 1.0 2/02/2008                                         *
\*****************************************************************/

#include <stdio.h>

int main( void )
{
  int j, cont = 0;

  for (j = 0; j < 32; j++)
    if (0xDB29 & (1L << j))
      ++cont;
  printf("num. bits: %d\n",cont);

  return 0;
}
```

```
num. bits: 9
```

Operadores para asignación de bits

Operador	Símbolo	Sintaxis	Operación
asig. despl. der.	>>=	x >>= y	asigna x>>y a x
asig. despl. izq.	<<=	x <<= y	asigna x<<y a x
asigna - AND	&=	x &= y	asigna x & y a x
asigna - OR	|=	x |= y	asigna x | y a x
asigna - XOR	^=	x ^= y	asigna x ^ y a x

Operador condicional (ternario)

Operador	Símbolo	Sintaxis	Operación
condicional	? :	a ? b : c	si a no es cero devuelve b, sino c

Equivalencia:

```
if (x<y)                    z = (x < y) ? x : y ;
        z=x;
else
        z=y;
```

Ejm: `x = (a > 0) ? a : −a ;  /* valor absoluto  de a */`

Operador paréntesis

- Permite agrupar los operandos y operadores que aparecen dentro de los paréntesis en primer lugar, alterando la precedencia natural de las operaciones.

Operador	Símbolo	Sintaxis	Operación
paréntesis	(…)	(a+b)/(c*d)	Altera la precedencia

Operador cast

- Permite convertir el valor resultante de una expersión a un tipo de datos diferente.

Operador	Símbolo	Sintaxis	Operación
cast	(tipo)	(tipo) expr.	Convierte expr. a tipo

- Ejm: (float) 3 / 2

Operador sizeof

- La llamada sizeof() se utiliza para determinar el número de bytes que ocupa una variable o un tipo

Operador	Símbolo	Sintaxis	Operación
sizeof	sizeof	sizeof (t) sizeof (x)	Devuelve el tamaño, en bytes, del tipo de dato t o expresión x.

- Ejm:

```
int a = 5;
printf ("bytes de a: %d\n",sizeof(a));
printf ("bytes de double: %d\n", sizeof(double));
```

```
4
8
```

Operadores para manejo de memoria

- Aplicables en variables de tipo arrays y punteros

Operador	Símbolo	Sintaxis	Operación
dirección de	&	&x	dirección de x
contenido direcc.	*	*a	valor en direc. a
contenido array	[ ]	x[5]	valor elem. x[5]
Punto	.	x.y	valor miembro y de la estructura x
puntero estruc.	->	p -> y	valor miembro y de la estruct. apunt.por p