Cap. 2 - Introducción a la programación en C

Esquema

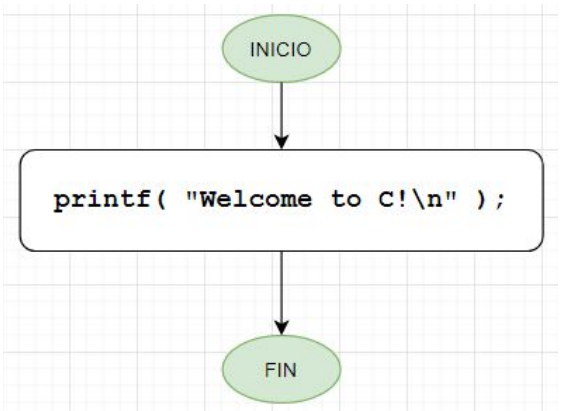
2.1 Introducción

- Lenguaje de programación C
  - **Un enfoque estructurado y disciplinado para el diseño del programa**
- Programación estructurada
  - Introducido en los clases 3 y 4
  - Utilizado a lo largo del curso

2.2 Un Simple Programa en C: Imprimir una línea de Texto

```
1   /* Fig. 2.1: fig02_01.c
2      Un pimer programa en C */
3   #include <stdio.h>
4
5   int main()
6   {
7      printf( "Welcome to C!\n" );
8
9      return 0;
10  }
```

```
Welcome to C!
```



- Comentario
  - El texto rodeado por /* y */ es ignorado por la computadora
  - Se utiliza para describir el programa
- **`#include <stdio.h>`**
  - Directivas al Preprocesador - le dice a la computadora que cargue el contenido de un cierto archivo
  - **`<stdio.h>`** permite operaciones estándar de entrada/salida

2.2 Un Simple Programa en C: Imprimir una línea de Texto (II)

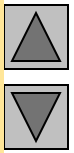
- **`int main()`**
  - Los programas en C contienen one o más funciones, exactamente uno de los cuales debe ser **`main`**
  - Los paréntesis “( )”utilizados indican una función
  - **`int`** significa que main “retorna” un valor entero
  - Los corchetes indican un bloqueo
    - Los cuerpos de todas las funciones deben estar contenidos en corchetes

2.2 Un Simple Programa en C: Imprimir una línea de Texto (III)

- **`printf( "Welcome to C!\n" );`**
  - Instruye a la computadora para que realice una acción
    - Específicamente, imprime cadenas de caracteres dentro de las comillas
  - Toda la línea se llama declaración
    - Todas las declaraciones deben terminar con un punto y coma
  - **`\`** - carácter escape
    - Indicate que **`printf`** debería hacer algo fuera de lo ordinario
    - **`\n`** es un carácter de nueva línea

2.2 Un Simple Programa en C: Imprimir una línea de Texto (IV)

- **`return 0;`**
  - Una forma de salir de una función
  - **`return 0`**, en este caso, significa que el programa termina normalmente
- Corchete derecho **`}`**
  - Indica fin que la función **`main`** ha sido alcanzado
- Enlazador
  - Cuando una función es invocada, el enlazador lo ubica en la librería
  - Lo inserta en el programa objeto
  - Si el nombre de la función está mal escrito, el enlazador detectará el error porque no puede encontrar la función en la biblioteca



Outline

```c
/* Fig. 2.5: fig02_05.c
   Programa de Suma */
#include <stdio.h>

int main()
{
   int integer1, integer2, sum;        /* declaración */

   printf( "Ingrese primer entero \n" );  /* Indicar */
   scanf( "%d", &integer1 );              /* lee un entero */
   printf( "Ingrese segundo enterio\n" ); /* Indicar */
   scanf( "%d", &integer2 );              /* lee un entero */
   sum = integer1 + integer2;             /* asigna la suma */
   printf( "La suma es %d\n", sum );   /* imprime la suma */

   return 0; /*indica que el programa termina correctamente */
}
```

1. Inicia variables

2. entradas

2.1 Suma

3. Imprime

Salida del Programa

```
Ingrese primer entero
45
Ingrese segundo entero
72
La suma es 117
```

2.3 Otro simple Programa en C: Sumar Dos Enteros

- Como antes.
  - Comentarios, **`#include <stdio.h>`** y **`main`**
- **`int integer1, integer2, sum;`**
  - Declaración de variables
    - Variables: lugares en la memoria donde un valor puede ser almacenado
  - **`int`** significa que las variables pueden contener números enteros (**`-1`**, **`3`**, **`0`**, **`47`**)
  - **`integer1`**, **`integer2`**, **`sum`** - nombre de variables (identificadores)
    - Identificadores: consiste de letras, dígitos (no puede iniciar con un dígito), y subraya, distingue entre mayúsculas y minúsculas
  - Las declaraciones aparecen antes que las declaraciones ejecutables
    - Si no, error de sintaxis (compilación)

2.3 Otro simple Programa en C: Sumar Dos Enteros (II)

- `scanf( "%d", &integer1 );`
  - Obtiene el valor del usuario
    - **`scanf`** usa entrada estandar (usualmente teclado)
  - Este **`scanf`** tiene dos argumentos
    - `%d` - indica que el dato debe ser un entero decimal
    - **`&integer1`** - ubicación en la memoria para almacenar la variable
    - `&` es confuso en principio - solo recuerde incluirlo con el nombre de la variables en **`scanf`** declaraciones
      - Esto será discutido más adelante
  - Los usuarios responden a **`scanf`** escribiendo en número, entonces presionando la tecla enter (return)

2.3 Otro simple Programa en C: Sumar Dos Enteros (III)

- **=** (operador de asignación)
  - Asigna un valor a una variable
  - Operador binario (tiene dos operandos)

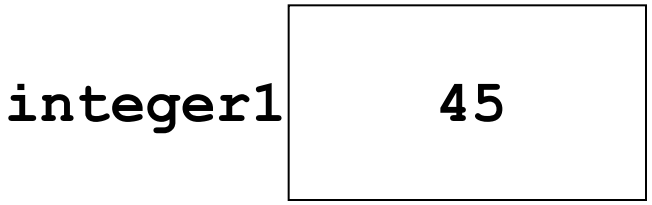
    ```
    sum = variable1 + variable2;
    ```

    sum consigue **variable1 + variable2;**
  - Valor de recepción variable a la izquierda
- **printf("Sum is %d\n", sum);**
  - Similar a **scanf** - **%d** significa que un entero decimal será impreso
    - **sum** especifica que número entero será impreso
  - Cálculos pueden ser realizados dentro de la declaración **printf**

    ```
    printf( "Sum is %d\n", integer1 + integer2 );
    ```


2.4 Conceptos de Memoria

- Variables
  - Los nombres de las variables corresponden a lugares de la memoria del ordenador.
  - Cada variable tiene un ***nombre***, un ***tipo***, un ***tamaño*** y un ***valor***.
  - Cada vez que se coloca un nuevo valor en una variable (a través de **`scanf`**, por ejemplo), lo reemplaza (y destruye) valores previos
  - La lectura de las variables de la memoria no las cambia
- Una representación visual



2.5 Aritmetica

- Los cálculos aritméticos se usan en la mayoría de los programas
  - Usar `*` para multiplicación y `/` para división
  - La división entera trunca el resto

    `7 / 5` evalua a `1`
  - El operador del módulo devuelve el resto

    `7 % 5` evalua a `2`
- Precedencia del operador
  - Algunos operadores aritméticos actúan antes que otros (esto es, multiplicación antes de la suma)
    - Usar paréntesis cuando sea necesario
  - Ejemplo: Encuentra el promedio de tres variables `a`, `b` y `c`
    - NO use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3`

2.5 Aritmetica (II)

- Operadores Aritméticos:

Operación	Operador Aritmético	Expresión Algebraica	Expresión en C
Adición	+	***f + 7***	f + 7
Substracción	-	***p - c***	p - c
Multiplicación	*	***bm***	b * m
División	/	***x / y***	x / y
Módulo	%	***r mod s***	r % s

- Regla de precedencia de operadores

Operador(es)	Operacion(es)	Orden de evaluación (precedencia)
( )	Parentesis	Evaluado Primero. Si los paréntesis son anidados, la expresión en el par más interno es primero evaluado.Si hay varios pares de paréntesis “en el mismo nivel”, son evaluados de izquierda a derecha.
*, /, %	Mutiplicación, División, Módulo	Evaluado segundo. Si hay varios, ellos son evaluados de izquierda a derecha.
+, -	Adición, Subtracción	Evaluado ultimo. Si hay varios, ellos son evaluados de izquierda a derecha.

2.6 Toma de decisiones: Igualdad y operadores relacionales

- Declaraciones ejecutables
  - Realiza acciones (calculos, entrada/salida de datos)
  - Realiza decisiones
    - Puede querer imprimir "aprobado" o "reprobado" dado el valor de la nota de un examen
- `if` estructura de control
  - Versión simple en esta sección, más detalles más adelante
  - Si la condición es verdadera, entonces el cuerpo de la expresión **`if`** se ejecuta
    - `0` es falso, distinto de cero es verdadero
  - El control siempre se reanuda después de la estructura `if`
- Palabras clave
  - Palabras especiales reservadas para C
  - No pueden utilizarse como identificadores o nombres de variables

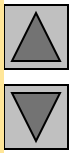
2.6 Toma de decisiones: Igualdad y operadores relacionales (II)

Operador estándar de igualdad algebraica u operador relacional	Igualdad en C u Operador Relacional	Ejemplo de condición en C	Significado de condición en C
Operadores relacional			
>	>	x > y	x es mayor que y
<	<	x < y	x es menor que y
≥	>=	x >= y	x es mayor o igual que y
≤	<=	x <= y	x es menor o igual que y
Operadores de igualdad			
=	==	x == y	x es igual a y
≠	!=	x != y	x no es igual a y

2.6 Toma de decisiones: Igualdad y operadores relacionales (III)

Palabras claves			
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

```c
/* Fig. 2.13: fig02_13.c
   Usando declaración if, operador
   relacional, y operador de igualdad */
#include <stdio.h>

int main()
{
   int num1, num2;

   printf( "Ingrese dos enteros, y Yo te diré \n" );
   printf( "la relación que satisfacen: " );
   scanf( "%d%d", &num1, &num2  );   /* lee dos enteros */

   if ( num1 == num2 )
      printf( "%d es igual a %d\n", num1, num2 );

   if ( num1 != num2 )
      printf( "%d no es igual a %d\n", num1, num2 );

   if ( num1 < num2 )
      printf( "%d es menor que %d\n", num1, num2 );

   if ( num1 > num2 )
      printf( "%d es mayor que %d\n", num1, num2 );

   if ( num1 <= num2 )
      printf( "%d es menor o igual que %d\n",
              num1, num2 );
```



Outline

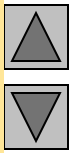
1. Declarar las variables

2. Entrada

2.1 declaración `if`

3. Imprime

```
29
30      if ( num1 >= num2 )
31         printf( "%d es mayor o igual que %d\n",
32                 num1, num2 );
33
34      return 0;   /* indican que el programa terminó con éxito */
35   }
```



Outline

3.1 Salida `main`

Resultado del Programa

```
Ingrese dos enteros, y Te dire
la relación que satisfacen: 3 7
3 no es igual a 7
3 es menor que 7
3 es menor o igual a 7
```

```
Ingrese dos enteros, y Te dire
la relación que satisfacen: 22 12
22 no es igual a 12
22 es mayor que 12
22 es mayor o igual a 12
```