UNT – FACEyT - DEEC

Programación I

Año 2016

Introducción

Docentes

Profesor:

Ing. Jorge Steifensand

Auxiliares :

- Ing. José Younes
- Ing. María Fernanda Guzmán
- Ing. Mariana Sánchez
- Ing. Teresa Cardozo

Introducción

Carreras

- Ingeniería Eléctrica
- Ingeniería Electrónica
- Ingeniería en Computación

Introducción

Condiciones para cursar

- Cálculo I: ***Aprobada***
- Álgebra y Geometría Analítica: ***Aprobada***
- Informática: ***Regular***

Condiciones para rendir

- Informática : ***Aprobada***
- Programación I : ***Regular***

Introducción

Régimen de cursado

Teoría (2 horas semanales):

- Jueves de hs. 08:00 a 10:00
- Anfiteatro B1

Práctica (4 horas semanales):

- Asistencia obligatoria
- Los alumnos deben elegir 2 turnos de 2 hs. c/u
- Gabinete 1 DEEC (1-3-08)

  (Block 1 – Piso 3 – Aula 8)

Introducción

Régimen de cursado (cont.)

Tres (3) evaluaciones de práctica en Gabinete.

- Las evaluaciones se aprueban con nota 6 (seis) o más.
- Para regularizar:
  - ✓ se deben aprobar las 3 evaluaciones.
  - ✓ tener porcentaje de asistencia mínimo del 75% en las clases prácticas.
- Las evaluaciones desaprobadas se recuperan:
  - ✓ Cada parcial en forma individual antes del receso invernal.
  - ✓ Un integral luego del receso.

Introducción

Sitio Web de la asignatura:

`catedras.facet.unt.edu.ar/prog1`

Para acceder al Material Didáctico del sitio:

Alumnos → Material didáctico

Password: **`ArrayList`**

Aclaración:

- Los códigos de los ejemplos pueden contener errores. Estos se introducen de exprofeso para ser discutidos en clase.
- Aquellos alumnos que no asistan a clase de teoría presten atención a los errores de compilación y/o ejecución de los ejemplos.

Introducción

Bibliografía

- C. Manual de referencia, 4ta. edición, Herbert Schildt, McGraw-Hill
- Cómo programar en C/C++, H.M. Deitel y P.J. Deitel, Prentice Hall
- Programación en C, Joyanes Aguilar y Zahonero Martínez, McGraw-Hill

Breve historia del lenguaje C

- 1966 → **BCPL**: *Basic Combined Programming Language* (Lenguaje de Programación Básico Combinado). Martin Richards, Cambridge.
- 1969 → **B**: Dennis Ritchie y Ken Thompson. Laboratorios Bell. Reescribir el UNIX que estaba en ensamblador.
- 1972 → **C:** Dennis Ritchie y Brian Kernighan
- 1989 - 1990 → ANSI C90 – ISO/IEC 9899:1990
- 2011 → ISO/IEC 9899:2011

Introducción

Tarea	Herramienta	Se obtiene
Edición	Editor de texto	<nom_arch>.c (código fuente)
Compilación	Compilador (se compila el código fuente): para DOS / Windows ...>**gcc nom_arch.c** →	(archivo ejecutable) **a.exe**
	...>**gcc nom_arch.c -o nom_arch** →	**nom_arch.exe**
	para Linux / UNIX ...**$gcc nom_arch.c** →	**a.out**
	...**$gcc nom_arch.c -o nom_arch** →	**nom_arch**
Ejecución	Archivo ejecutable DOS / Windows (según el resultado de la compilación) ...> **a.exe** (nom_arch.exe)	Ejecución del programa • Ingreso de datos • Procesamiento • Salida de datos
	Linux / UNIX ...**$./a.out** (./nom_arch)	

Unidad 1: Estructura de un programa en C

1.- /* Comentarios: Los comentarios se escriben entre
/* en varias líneas */
o con //..... hasta el final de la línea
*/

2.- // Directivas al compilador:

```
#include <librería.h>
```

3.- // Declaraciones globales (funciones)

4.- // Función principal

```
int main (int  argc, char  *argv[])
{
    /*declaraciones locales y sentencias*/
return 0;
}
```

5.- // Definición de las funciones declaradas en el punto 3.-

Unidad 1: Tipos primitivos de datos

Tipo	bits	rango
char	8	-128 a127
int/long int	32	-2,147,483,648 a 2,147,483,647
float	32	± 3.4 E-38 a ± 3.4 E+38 (7 dígitos)
double	64	±1.7E-308 a ±1.7E+308 (15 dígitos)

El Lenguaje C no tiene datos tipo "*booleanos*":

- toma como *"verdadero"* cualquier valor distinto que cero,
- Una expresión devuelve **1** como valor verdadero y **0** como falso.

Unidad 1: Identificadores

- Nombre que se les da a las constantes, variables y/o funciones.
- Está formado por una secuencia de caracteres alfanuméricos (conjunto 1 de caracteres ASCII).
- NO puede tener espacios ni símbolos especiales.
- DEBE comenzar con un caracter alfabético.
- C es sensible al tipo de letra, diferencia entre mayúsculas y minúsculas.
- Se recomienda:
  - ✓ usar letras minúsculas para variables y funciones,
  - ✓ letras mayúsculas para las constantes.

Unidad 1: Tabla de caracteres ASCII

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US		!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	E	D
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Caracteres imprimibles

Caracteres no imprimibles

Unidad 1: Identificadores (cont.)

Ejemplos de identificadores:

car	→	Válido
día	→	Identificador no válido (símbolo í)
cont_1	→	Válido
30dias	→	Identificador no válido (el primer caracter numérico)
suma	→	Válido
cont 2	→	Identificador no válido (tiene espacio en blanco)

Unidad 1: Variables

- Una variable es un símbolo que se utiliza para representar un dato.
- En informática representa el domicilio de memoria donde se guarda el dato asignado.
- En el lenguaje C la variables deben ser declaradas antes de ser usadas.

Declaración:

`tipo_de_dato` nomb_de_la_var1 [,nombre_de_la_var2, ...] ;

Ejemplos:

```
char    car;
int     cont_1, suma;
float   promedio;
```

Unidad 1: Constantes (1/4)

- Una constante es un símbolo que se utiliza para representar un dato cuyo valor no puede modificarse durante la ejecución de un programa.
- Las constantes pueden ser:

1. literales,
2. definidas o simbólicas, o
3. declaradas.

Unidad 1: Constantes (2/4)

1. Literales:
   - Son valores de cualquier tipo que se utilizan directamente.
   - No se declaran ya que no tienen nombre.

Ejemplos:

'm'

7

12.098

"clases"

Unidad 1: Constantes (3/4)

2. Definidas o simbólicas:
   - Se declaran e inicializan en la zona de declaraciones globales
   - Se recomienda **NO utilizar constantes simbólicas**. La razón es que, para este tipo de constantes, el compilador no analiza compatibilidad en cuanto tipos de datos y se pueden producir errores difíciles de detectar.

Ejemplos:

```
#define R 0.082
#define rho_Cu 0.0172
```

Unidad 1: Constantes (4/4)

3. Declaradas:
   - Se declaran como una variable precedidas de la palabra reservada **`const`**.
   - Deben inicializarse cuando se declaran.

Declaración:

`const` *`tipo_de_dato`* **nomb_de_la_const** = valor;

Ejemplos:

```
const    int   DIAS = 12;
const    float R = 0.082;
```

Unidad 1: Asignación

- Es la acción mediante la cual una variable toma un valor determinado.
- El símbolo **=** representa la asignación en lenguaje C.

Sintaxis:

variable = expresión;

Ejemplos de asignación:

```
car       = 'r';
cont_1    = 16;
promedio  = cont_1 / 5;
```

Se puede asignar valor a una variable cuando se declara:

```
int suma = 0;
```

Unidad 1: Operadores (1/4)

Operadores aritméticos

suma	resta	multiplicación	división	módulo
+	-	*	/	%

Operadores de relación

igual que	distinto que	mayor que	mayor o igual que	menor que	menor que
==	!=	>	>=	<	<=

Operadores lógicos

NO (not)	O (or)	Y (and)
!	||	&&

Unidad 1: Operadores (2/4)

Operadores unarios

se aplican sólo a un operando

dirección	indirección	incremento unitario	decremento unitario
&	*	++	- -

&x → dirección de la variable ***x***

****p*** → contenido de la variable a la que apunta ***p***

++x → incrementa el valor y luego lo utiliza

x++ → utiliza el valor y luego lo incrementa

- -x → decrementa el valor y luego lo utiliza

x- - → utiliza el valor y luego lo decrementa

Unidad 1: Operadores (3/4)

Operador condicional

Este operador evalúa la expresión, en función del resultado de esta ejecuta una de las dos instrucciones.

Su sintaxis es:

(expresión) ? instrucción_V : instrucción_F;

Si el resultado de expresión es:

- verdadero se ejecuta **instrucción_V**,
- falso se ejecuta **instrucción_F**.

Unidad 1: Operadores (4/4)

Abreviatura de operaciones

En lenguaje C la asignación:

variable = variable OPERADOR *expresión*

es equivalente a:

variable OPERADOR **=** *expresión*

Operador	Forma abreviada	Forma no abreviada
+=	z += (x + y)	z = z + (x + y)
-=	z -= (x + y)	z = z – (x + y)
*=	z *= (x + y)	z = z * (x + y)
/=	z /= (x + y)	z = z / (x + y)
%=	z %= (x + y)	z = z % (x + y)

Unidad 1: Funciones de entrada/salida (1/7)

Ingreso de datos y salida de resultados

- El dispositivo estándar, en el lenguaje C, por donde se ingresan los datos es el teclado.
- El correspondiente a la salida de resultados es el monitor.

Tanto como para ingresar datos por teclado como para mostrar información por monitor existen una serie de funciones estándares definidas en un archivo cabecera o "*header*" llamado:

`stdio.h`

Este archivo debe ser incluido en la compilación para poder hacer uso de las funciones definidas en él.

Unidad 1: Funciones de entrada/salida (2/7)

Salida de resultados por monitor o pantalla (1/4)

La función más utilizada es:

printf *("cadena de control" [, lista de argumentos]);*

- Los corchetes indican "opcional",
- la función puede o no utilizar argumentos,
- en caso de no utilizar argumentos su función es mostrar mensajes.

Unidad 1: Funciones de entrada/salida (3/7)

Salida de resultados (2/4)

La "cadena de control" puede contener: texto (mensaje), secuencias de escape y/o especificadores de tipos de datos.

Secuencias de escape	
\n	Salto de línea
\t	Tabulación horizontal
\a	Señal sonora
\b	Retroceso
\r	Retorno de carro
\\	Barra invertida
\'	Comilla simple
\"	Comilla doble
\?	Interrogación
\0	Caracter nulo

Especificador de tipo de dato	
%c	caracter (char)
%i	entero (int)
%d	entero (formato decimal)
%f	punto decimal flotante
%e	forma exponencial (1.23e+2)
%E	forma exponencial (1.23E+2)
%s	cadena de caracteres (string)
%p	puntero (dirección de memoria)

Unidad 1: Funciones de entrada/salida (4/7)

Salida de resultados (3/4)

Salidas con formato:

El formato de salida tiene el siguiente prototipo:

%[bandera][ancho][.precisión] especificador

donde

- bandera:

-	Justifica a la izquierda
+	Inserta signo + ó -
0	Completa con ceros a la izquierda

Unidad 1: Funciones de entrada/salida (5/7)

Salida de resultados (4/4)

- ancho:
  - ✓ Es un valor entero.
  - ✓ Indica la cantidad de espacios que se utilizarán para representar los valores.
  - ✓ Si la cantidad de espacios que ocupa un valor es superior al "ancho" lo mismo se representa.
  - ✓ En los valores con punto decimal en el ancho se incluyen el punto decimal y los dígitos decimales.
- precisión:
  - ✓ Sólo para valores con punto decimal flotante.
  - ✓ Indica la cantidad de dígitos decimales que se mostrarán.

Unidad 1: Funciones de entrada/salida (6/7)

Ingreso datos por teclado (1/2)

La función más utilizada es:

scanf *("cadena de control", lista de direcciones de memoria);*

- *cadena de control*

  Contiene el especificador del tipo de dato a leer.

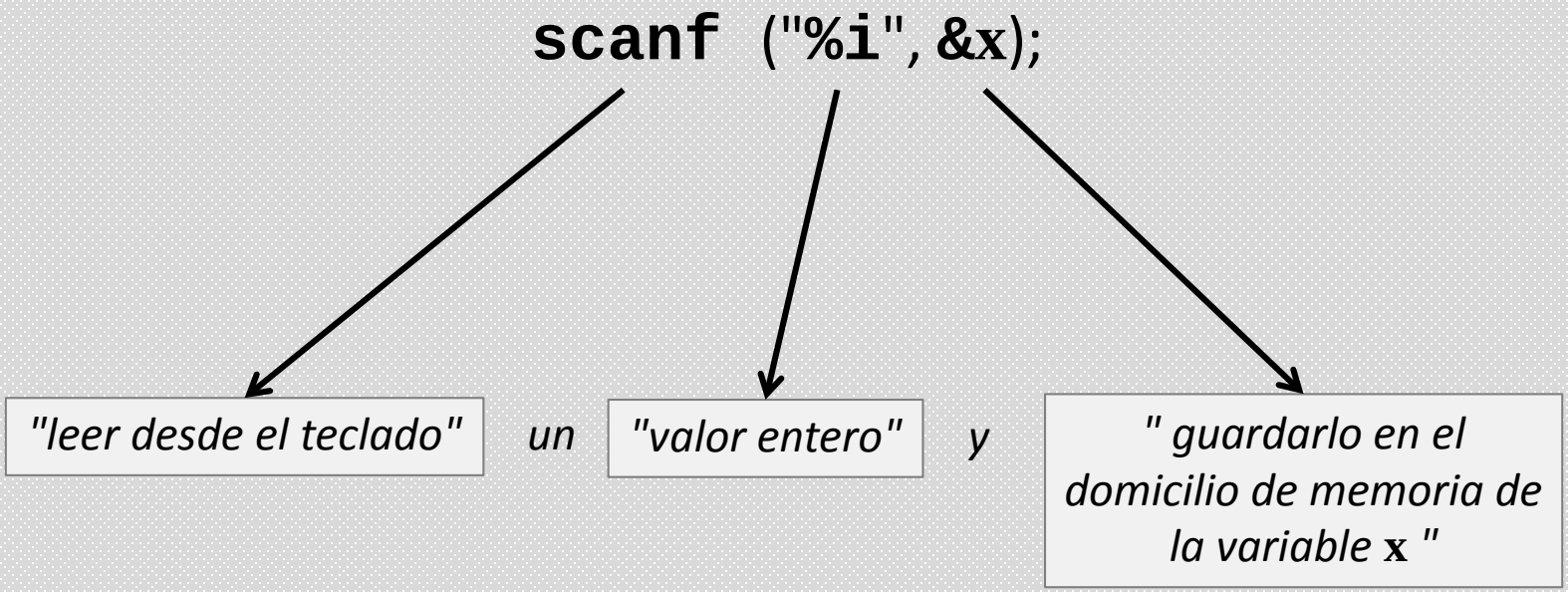
- *lista de direcciones de memoria*

  Direcciones de memoria correspondientes a las variables a las que se les asignará el valor leído.

Unidad 1: Funciones de entrada/salida (7/7)

Ingreso datos por teclado (2/2)

Ejemplo de utilización de la función **`scanf`** ():



Unidad 1: Estructuras de control (1/10)

Selección simple

```
if (expresión)
        {
            // bloque de código
        }
```

Se evalúa la **expresión**, si el resultado es **verdadero** (distinto que cero) se ejecutan la/s sentencia/s del bloque de código. Caso contrario el flujo del programa continúa con la sentencia siguiente a esta estructura.

Si el bloque de código tiene una sola sentencia se pueden omitir las llaves

Unidad 1: Estructuras de control (2/10)

Selección doble

```
if (expresión)
    {
        // bloque de código V
    }
else
    {
        // bloque de código F
    }
```

Si la **expresión** arroja resultado verdadero se ejecutan las sentencias del *"bloque de código V"*. Caso contrario se ejecutan las sentencias del *"bloque de código F"*.

Unidad 1: Estructuras de control (3/10)

Selección múltiple (1/2)

```
switch (selector)
        {
                case valor_1:
                        //bloque de código_1
                        break;
                case valor_2:
                        //bloque de código_2
                        break;
                . . .
                case valor_N:
                        //bloque de código_N
                        break;
                [default :
                        //bloque de código_default
                        break;]
        }
```

Unidad 1: Estructuras de control (4/10)

Selección múltiple (2/2)

selector:

Es una variable del tipo ordinal (entero o caracter). Según el valor de esta variable se ejecutará uno de los bloques de código (**`case`**).

case:

Corresponden a las distintas opciones de la estructura **`switch`**. El bloque de código de cada **`case`** debe finalizar "siempre" con la sentencia **`break`**.

Si un bloque **`case`** no finaliza con la sentencia **`break`**, se ejecutarán todos los bloques **`case`** subsiguientes, hasta encontrar una sentencia **`break`**.

default:

Este bloque de código es opcional. Se ejecuta si la variable selector no tomó ningún valor de las opciones establecidas.

Unidad 1: Estructuras de control (5/10)

Repetición: estructura **`while ...`** (1/2)

```
while (expresión)
    {

        // bloque de código

    }
```

Unidad 1: Estructuras de control (6/10)

Repetición: `while ...` (2/2)

Si la **expresión** arroja resultado verdadero se ejecutan las sentencias del *"bloque de código"*. Caso contrario el flujo del programa continúa con la sentencia siguiente del programa.

La **expresión** contiene una variable denominada "variable de control". A esta variable se le debe asignar un valor antes del inicio de la estructura.

En el bloque de código de la estructura `while ...` debe existir una sentencia donde se modifique el valor de la variable de control. Si esta no es modificada el ciclo será infinito.

Es posible que nunca se ejecute el bloque de código, ya que la expresión arroja el valor falso en la primera evaluación.

Unidad 1: Estructuras de control (7/10)

Repetición: Estructura **do ... while** (1/2)

...

```
do {

    // bloque de código

} while (expresión);
```

Unidad 1: Estructuras de control (8/10)

Repetición: Estructura `do ... while ...` (2/2)

En esta estructura siempre se ejecutan, al menos una vez, las sentencias del bloque de código. Luego de la ejecución evalúa la **expresión**, mientras esta arroje el valor verdadero volverá a ejecutar el bloque de código. Si el valor devuelto por la **expresión** es falso el flujo del programa continúa con la sentencia siguiente del programa.

La **expresión** contiene una variable denominada "variable de control". A diferencia de la estructura `while ...` esta variable puede ser inicializada antes del inicio de la estructura o dentro del bloque de código. Debe existir una sentencia donde se modifique el valor de la variable de control. Si esta no es modificada el ciclo será infinito.

Unidad 1: Estructuras de control (9/10)

Repetición: Estructura `for ( ; ; ) ...`

```
for (valor inicial; condición; incremento)
{
        // bloque de código
}
```

Esta estructura es similar a la estructura while ... , sólo que inicialización, condición (expresión) y modificación de la variable de control se realizan en una sola línea, separadas por punto y coma(`;`). Esta estructura tiene variantes que se analizarán en clases próximas.

Unidad 1: Estructuras de control (10/10)

Sentencias:

- **`continue`**
- **`break`**

`continue`:

es una sentencia que interrumpe la ejecución de un ciclo en una estructura de repetición, forzando a comenzar el ciclo siguiente.

`break`:

es una sentencia de ruptura de ejecución en una estructura. Transfiere la ejecución a la estructura siguiente. Su uso es válido tanto en estructuras de repetición como de selección.

Unidad 1: Punteros (1/3)

- Un puntero es una variable cuyo contenido es la dirección de memoria de otra variable o constante.
- Se declara como todas las variables precediendo al identificador del símbolo asterisco (*).
- El puntero deber ser del mismo tipo que la variable o constante apuntada.

Sintxis de la declaración:

***tipo_de_dato** *nombre_del_puntero;*

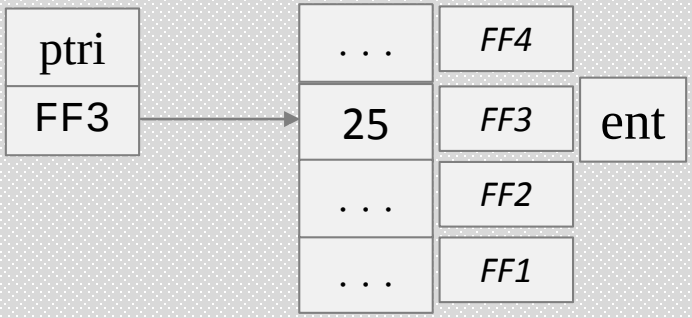
Operadores:

- dirección: ***&***
- indirección: *******

Unidad 1: Punteros (2/3)

Ejemplo:

```
int     ent, *ptri;
float   real, *ptrf;
char    car, *ptrc;
ptri = &ent;
//a ptri se le asigna la dirección ent
*ptri = 25;
//por indirección se asigna a ent el valor 25
ptrf = &ent;
// INSTRUCCIÓN INVÁLIDA,
//ptrf debe apuntar a un  float  y se le asigna dirección de int
ptrf = &real;
```



Unidad 1: Punteros (3/3)

```
int   ent, *ptri;
float real, *ptrf;
char  car, *ptrc;

ptri = &ent;
ptrf = &real;
ptrc = &car;

//Incremento
//de punteros

ptri++;
ptrf++;
ptrc++;
```