	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Marco Antonio Martínez Quintana

Asignatura: Fundamentos de Programación

Grupo: 3

No de Práctica(s): 7

Integrante(s): Ramírez Pérez Daniela Itzel

No. de Equipo de cómputo empleado: No aplica

No. de Lista o Brigada: 36

Semestre: 1er

Fecha de entrega: Viernes 27 de Noviembre

Observaciones:

CALIFICACIÓN: _____

Objetivo:

Elaborar programas en lenguaje C utilizando las instrucciones de control de tipo secuencia, para realizar la declaración de variables de diferentes tipos de datos, así como efectuar llamadas a funciones externas de entrada y salida para asignar y mostrar valores de variables y expresiones.

Introducción

Cuando a un problema se le ha diseñado un algoritmo que lo resuelva y que se haya representado el algoritmo de manera gráfica o escrita se puede proceder a la etapa de codificación.

La codificación se puede realizar en cualquier lenguaje de programación estructurada, como lo son Pascal, Python, Fortran, PHP, C.

Dentro del ciclo de vida del software, la implementación de un algoritmo se encuentra dentro de la etapa de codificación del problema. Esta etapa va muy unida a la etapa de pruebas:

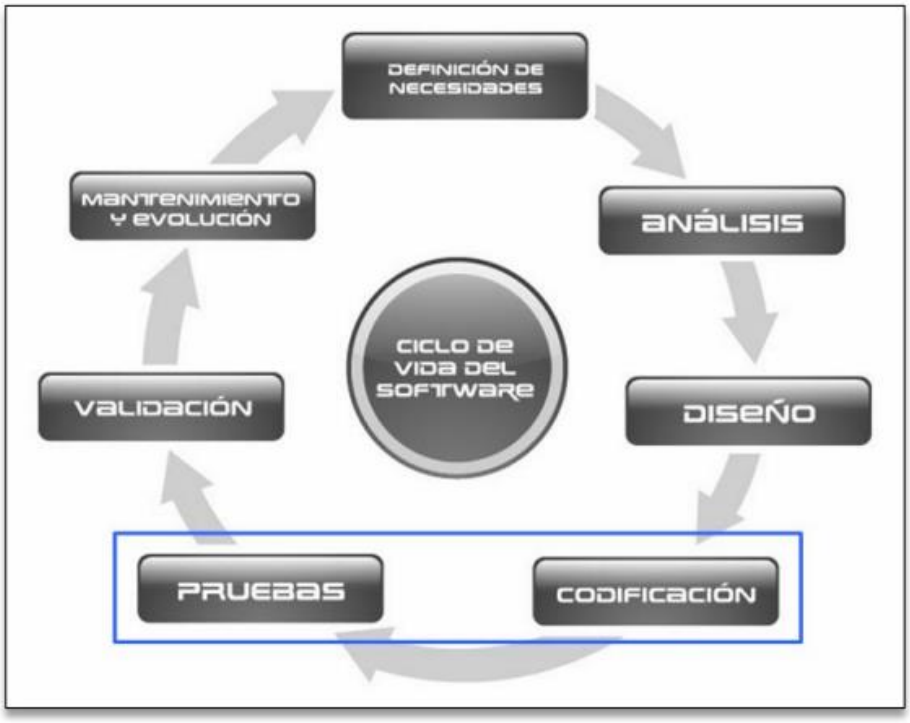

Figura 1: Ciclo de vida del software, resaltando las etapas de *códificación* y *pruebas*, las cuales se cubrirán en esta práctica.

Ejecución del lenguaje C

Al momento de ejecutar un programa objeto (código binario), se ejecutarán únicamente las instrucciones que estén definidas dentro de la función principal. La función principal puede contener sentencias, estructuras de control y comentarios. Dentro de las sentencias se encuentran la declaración y/o asignación de variables, la realización de operaciones básicas, y las llamadas a funciones.

Comentarios

En C existen dos tipos de comentarios:

- *El comentario por línea* inicia cuando se insertan los símbolos '//' y termina con el salto de línea.
- *El comentario por bloque* inicia cuando se insertan los símbolos '/*' y termina cuando se encuentran los símbolos '*/'.

Ejemplo

```
// Comentario por línea

/* Comentario por bloque
 que puede ocupar
 varios renglones */
```

Declaración de variables

Para declarar variables en C se sigue la siguiente sintaxis:

[modificadores] tipoDeDato identificador [= valor];

Esto permite que la variable tenga diferentes modificadores y que se deba declarar el tipo de dato que puede contener la variable.

También es posible declarar varios identificadores de un mismo tipo de dato e inicializarlos en el mismo renglón, lo único que se tiene que hacer es separar cada identificador por comas.

tipoDeDato identificador1[= valor], identificador2[= valor];

Tipos de datos

Los tipos de datos básicos en C son:

- **Caracteres**: codificación definida por la máquina.
- **Enteros**: números sin punto decimal.
- **Flotantes**: números reales de precisión normal.

Tipo	*Bits*	*Valor Mínimo*	*Valor Máximo*
signed char	8	-128	127
unsigned char	8	0	255
signed short	16	-32 768	32 767
unsigned short	16	0	65 535
signed int	32	-2,147,483,648	2 147 483 647
unsigned int	32	0	4 294 967 295
signed long	64	9 223 372 036 854 775 808	9 223 372 036 854 775 807
unsigned long	64	0	18 446 744 073 709 551 615
enum	16	-32 768	32 767

- **Dobles**: números reales de doble precisión

Las variables enteras que existen en lenguaje C son:

Si se omite el clasificador por defecto se considera 'signed'.

Las variables reales que existen en lenguaje C son:

Tipo	*Bits*	*Valor Mínimo*	*Valor Máximo*
float	32	3.4 E-38	3.4 E38
double	64	1.7 E-308	1.7 E308
long double	80	3.4 E-4932	3.4 E4932

Las variables reales siempre poseen signo.

Para poder acceder al valor de una variable se requiere especificar el tipo de dato. Los especificadores que tiene lenguaje C para los diferentes tipos de datos son:

Tipo de dato	*Especificador de formato*
Entero	%d, %i, %ld, %li, %o, %x
Flotante	%f, %lf, %e, %g
Carácter	%c, %d, %i, %o, %x
Cadena de caracteres	%s

El especificador de dato se usa para guardar o imprimir el valor de una variable.

Tipo de Valor entero	Especificador
valor entero en base 10	%d o %i
enteros largos	%ld ó %li
valor entero en base 8	%o
valor entero en base 16	%x.

Tipo de valor flotante	Especificador
valor real de precisión simple	%f
Valor real de doble precisión	%lf
notación científica	%e
redondea la parte fraccionaria a 3 dígitos significativos	%g

	Especificador
variable de tipo carácter	%c
valor del código ASCII del carácter en base 10	%i o %d
valor del código ASCII del carácter en base 8	%o
valor del código ASCII del carácter en base 16.	%x

	Especificador
cadenas de caracteres	%s.

Identificador

Un identificador es el nombre con el que se va a almacenar en memoria un tipo de dato. Los identificadores siguen las siguientes reglas:

- Debe iniciar con una letra [a-z].
- Puede contener letras [A-Z, a-z], números [0-9] y el carácter guión bajo (_).

Código declaración de variables

```
#include <stdio.h>
/*
 Este programa muestra la manera en la que se declaran y asignan variables
 de diferentes tipos: numéricas (enteras y reales) y caracteres.
*/
int main() {
 // Variables enteras
 short enteroNumero1 = 32768;
 signed int enteroNumero2 = 55;
 unsigned long enteroNumero3 = -789;
 char caracterA = 65; // Convierte el entero (ASCII) a carácter
 char caracterB = 'B';

 // Variables reales
 float puntoFlotanteNumero1 = 89.8;
 // La siguiente sentencia genera un error al compilar
 // porque los valores reales siempre llevan signo
 // unsigned double puntoFlotanteNumero2 = -238.2236;
 double puntoFlotanteNumero2 = -238.2236;
 return 0;
}
```

La biblioteca 'stdio.h' contiene diversas funciones para imprimir en la salida estándar (monitor) y para leer de la entrada estándar (teclado).

printf es una función para imprimir con formato, es decir, se tiene que especificar entre comillas el tipo de dato que se desea imprimir, también se puede combinar la impresión de un texto predeterminado:

```
printf("El valor de la variable real es: %lf", varReal);
```

scanf es una función que sirve para leer datos de la entrada estándar (teclado), cuando se especifica el tipo de dato entre comillas y en qué variable se quiere almacenar. Al nombre de la variable le antecede un ampersand (&), esto indica que el dato recibido se guardará en la localidad de memoria asignada a esa variable.

```
scanf ("%i", &varEntera);
```

Para imprimir con formato también se utilizan algunas secuencias de caracteres de escape, C maneja los siguientes:

- \a carácter de alarma
- \b retroceso
- \f avance de hoja
- \n salto de línea
- \r regreso de carro
- \t tabulador horizontal
- \v tabulador vertical
- ‘\0’ carácter nulo

Código almacenar e imprimir variables

```
#include <stdio.h>
/*
 Este programa muestra la manera en la que se declaran y asignan variables de diferentes tipos: numéricas (enteras y reales) y caracteres, así como la manera en la que se imprimen los diferentes tipos de datos.
*/
int main() {
 /* Es recomendable al inicio declarar
 todas las variables que se van a utilizar
 en el programa */
 // variables enteras
 int enteroNumero;
 char caracterA = 65; // Convierte el entero a carácter (ASCII)

 // Variable reales
 double puntoFlotanteNumero;

 // Asignar un valor del teclado a una variable
 printf("Escriba un valor entero: ");
 scanf("%d", &enteroNumero);
 printf("Escriba un valor real: ");
 scanf("%lf", &puntoFlotanteNumero);

 // Imprimir los valores con formato
 printf("\nImprimiendo las variables enteras:\a\n");
 printf("\tValor de enteroNumero = %i\n", enteroNumero);
 printf("\tValor de caracterA = %c\n",caracterA);
 printf("\tValor de puntoFlotanteNumero = %lf\n", puntoFlotanteNumero);

 printf("\nValor de enteroNumero en base 16 = %x\a\n", enteroNumero);
 printf("\tValor de caracterA en código hexadecimal = %i\n", enteroNumero);
 printf("\tValor de puntoFlotanteNumero en notación científica = %e\n",
 puntoFlotanteNumero);
 // La función getchar() espera un carácter para continuar la ejecución
 getchar();
 return 0;
}
```

Modificadores de alcance

Los modificadores que se pueden agregar al inicio de la declaración de variables son:

- El modificador *const* impide que una variable cambie su valor durante la ejecución del programa, Por convención, las constantes se escriben con mayúsculas y se deben inicializar al momento de declararse.

Ejemplo

```
const int NUM_MAX = 1000;
const double PI = 3,141592653589793 3;
```

- El modificador *static* indica que la variable permanece en memoria desde su creación y durante toda la ejecución del programa.

Ejemplo

```
static int num;
static float resultado = 0;
```

Código variables estáticas y constantes

```
#include <stdio.h>
/*
 Este programa muestra la manera en la que se declaran y asignan  las variables estáticas y las constantes.
*/
int main() {
 const int constante = 25;
 static char a = 'a';
 printf("Valor constante: %i\n", constante);
 printf("Valor estático: %c\n", a);
 // El valor de la variable declarada como constante no puede cambiar.
 // La siguiente línea genera un error al compilar si se quita el comentario:
 // constante = 30;
 // las variables estáticas sí pueden cambiar de valor
 a = 'b';
 printf("\nValor estático: %c\n", a);
 return 0;
}
```

Operadores

Los operadores aritméticos que maneja lenguaje C se describen en la siguiente tabla:

Operador	*Operación*	*Uso*	*Resultado*
+	Suma	125.78 + 62.5	188.28
-	Resta	65.3 - 32.33	32.97
*	Multiplicación	8.27 * 7	57.75
/	División	15 / 4	3.75
%	Módulo	4 % 2	0

Los operadores lógicos a nivel de bits que maneja el lenguaje C se describen en la siguiente tabla:

Operador	*Operación*	*Uso*	*Resultado*
>>	Corrimiento a la derecha	8 >> 2	2
<<	Corrimiento a la izquierda	8 << 1	16
&	Operador AND	5 & 4	4
|	Operador OR	3 | 2	3
~	Complemento ar-1	~2	1

Moldeo o cast

El resultado de una operación entre dos tipos de datos iguales puede dar como resultado un tipo de dato diferente, en esos casos es necesario moldear el resultado. A este proceso se le conoce como cast

Ejemplo:

```
// Si se tienen 2 enteros
int cinco = 5, dos = 2;
// La operación de división entre dos enteros
// genera un valor real, en este caso hay que
// moldear (cast) el resultado del lado derecho del
// igual para que corresponda con el lado izquierdo
// y se pueda asignar.
double res = (double)cinco/dos;
// Si no se hiciese el cast el resultado se truncaría.
```

Código operadores

```
#include <stdio.h>
/*
 Este programa muestra la manera en la que se realiza un moldeo o cast.
 También muestra como manipular números a nivel de bits:
 - Corrimiento de bits a la izquierda y a la derecha
 - Operador AND a nivel de bits
 - Operador OR a nivel de bits
*/
int main(){
 short ocho, cinco, cuatro, tres, dos, uno;

 // 8 en binario: 0000 0000 0000 1000
 ocho = 8;
 // 5 en binario: 0000 0000 0000 0101
 cinco = 5;
 // 4 en binario: 0000 0000 0000 0100
 cuatro = 4;
 // 3 en binario: 0000 0000 0000 0011
 tres = 3;
 // 2 en binario: 0000 0000 0000 0010
 dos = 2;
 // 1 en binario: 0000 0000 0000 0001
 uno = 1;
 printf("Operadores aritméticos\n");
double res = (double)cinco/dos; // Cast
 printf("5 / 2 = %lf\n",res);
printf("5 modulo 2 = %d\n",cinco%dos);
 printf("Operadores lógicos\n");
 printf("8 >> 2 = %d\n",ocho>>dos);
 printf("8 << 1 = %d\n",ocho<<1);
 printf("5 & 4 = %d\n",cinco&cuatro);
 printf("3 | 2 = %d\n",tres|dos);

 printf("\n");
 return 0;
}
```

Expresiones lógicas

Una expresión lógica puede tomar únicamente los valores verdadero o falso. Los operadores de relación permiten comparar elementos numéricos, alfanuméricos, constantes o variables.

Operador	*Operación*	*Uso*	*Resultado*
==	Igual que	'h' == 'H'	Falso
!=	Diferente a	'a' != 'b'	Verdadero
<	Menor que	7 < 15	Verdadero
>	Mayor que	11 > 22	Falso
< =	Menor o igual	15 <= 22	Verdadero
> =	Mayor o igual	20 >= 35	Falso

Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples.

Operador	*Operación*	*Uso*
!	No	! p
&&	Y	a > 0 && a < 11
| |	O	opc == 1 | | salir != 0

El operador ++ agrega una unidad (1) a su operando. Es posible manejar preincrementos

(++n) o posincrementos (n++).

El operador -- resta una unidad (1) a su operando. Se pueden manejar predecrementos (--n) o posdecrementos (n--).

Código expresiones lógicas

```
#include <stdio.h>
/*
 Este programa ejemplifica el manejo de operaciones relacionales y los
 operadores lógicos. También muestra la manera de incrementar o decrementar
 una variable y la diferencia entre hacerlo antes (pre) o después (pos).
*/
int main (){
 int num1, num2, res;
 char c1, c2;
 double equis = 5.5;
 num1 = 7;
 num2 = 15;
c1 = 'h';
 c2 = 'H';

 printf("Expresiones de relación\n");
 printf("¿ num1 es menor a num2 ? -> \t%d\n",num1<num2);
 printf("¿ c1 es igual a c2 ? -> \t%d\n",c1==c2);
 printf("¿ c1 es diferente a c2 ? -> \t%d\n",c1!=c2);

 printf("\nExpresiones lógicas\n");
 printf("¿ num1 es menor a num2 Y c1 es igual a 'h' ? -> \t%d\n",
 num1<num2 && c1 == 'h');
 printf("¿ c1 es igual a 's' O c2 es igual a 'H' ? -> \t%d\n",
 c1 == 's' || c2 == 'H');
 printf("¿ c1 es igual a 's' O c2 es igual a 'S' ? -> \t%d\n",
 c1 == 's' || c2 == 'S');

 printf("\nIncrementos y decrementos\n");
 printf("num1++ -> \t%d\n",num1++);
 printf("--num1 -> \t%d\n",--num1);
 printf("++equis -> \t%lf\n",++equis);

 return 0;
}
```

Depuración de programas

Cuando un programa falla y la información enviada por el compilador es muy general, se puede ejecutar el programa para saber, exactamente, dónde está fallando. Se revisará este tema en la guía práctica de estudio “Depuración de programas” para conocer las diferentes herramientas que nos ayudan a encontrar los errores de un programa.

Actividades:

calculadoraEnC.c | menorEdad.c | menu.c | areaPerimetro.c | almacenarV.c

```c
#include <stdio.h>
/*
 Este programa muestra la manera en la que se declaran y asignan variables de
 diferentes tipos: numéricas (enteras y reales) y caracteres, así como la manera
 en la que se imprimen los diferentes tipos de datos.
*/
int main() {
 /* Es recomendable al inicio declarar
 todas las variables que se van a utilizar
 en el programa */
 // variables enteras
 int enteroNumero;
 char caracterA = 65;
char au=163, ao=162, aa=160, ai=161;
 // Convierte el entero a carácter (ASCII)

 // Variable reales
 double puntoFlotanteNumero;

 // Asignar un valor del teclado a una variable
 printf("Escriba un valor entero: ");
 scanf("%d", &enteroNumero);
 printf("Escriba un valor real: ");
 scanf("%lf", &puntoFlotanteNumero);

 // Imprimir los valores con formato
 printf("\nImprimiendo las variables enteras:\a\n");
 printf("\tValor de enteroNumero = %i\n", enteroNumero);
 printf("\tValor de caracterA = %c\n",caracterA);
 printf("\tValor de puntoFlotanteNumero = %lf\n", puntoFlotanteNumero);

 printf("\nValor de enteroNumero en base 16 = %x\a\n", enteroNumero);
 printf("\tValor de caracterA en c%cdigo hexadecimal = %i\n",ao,enteroNumero);
 printf("\tValor de puntoFlotanteNumero en notaci%cn cient%cfica = %e\n",ao,ai,puntoFlotanteNumero);
 // La función getchar() espera un carácter para continuar la ejecución
 getchar();
 return 0;
}
```

```
C:\Users\twins\Desktop\Lenguaje C\Practica7>almacenarV.exe
Escriba un valor entero: 5
Escriba un valor real: 4

Imprimiendo las variables enteras:
        Valor de enteroNumero = 5
        Valor de caracterA = A
        Valor de puntoFlotanteNumero = 4.000000

Valor de enteroNumero en base 16 = 5
        Valor de caracterA en código hexadecimal = 5
        Valor de puntoFlotanteNumero en notación científica = 4.000000e+000
```

calculadoraEnC.c | menorEdad.c | menu.c | areaPerimetro.c | almacenarV.c | estYCons.c

```c
#include <stdio.h>
/*
 Este programa muestra la manera en la que se declaran y asignan  las variables estáticas y las constantes.
*/
int main() {
 const int constante = 25;
 static char a = 'a';
 char aa=160;
 printf("Valor constante: %i\n", constante);
 printf("Valor est%ctico: %c\n",aa, a);
 // El valor de la variable declarada como constante no puede cambiar.
 // La siguiente línea genera un error al compilar si se quita el comentario:
 // constante = 30;
 // las variables estáticas sí pueden cambiar de valor
 a = 'b';
 printf("\nValor est%ctico: %c\n",aa, a);
 return 0;
}

```

```
C:\Users\twins\Desktop\Lenguaje C\Practica7>estYCons.exe
Valor constante: 25
Valor estático: a

Valor estático: b
```

menu.c | areaPerimetro.c | almacenarV.c | estYCons.c | codigoOperadores.c

```c
#include <stdio.h>
/*
 Este programa muestra la manera en la que se realiza un moldeo o cast.
 También muestra como manipular números a nivel de bits:
 - Corrimiento de bits a la izquierda y a la derecha
 - Operador AND a nivel de bits
 - Operador OR a nivel de bits
*/
int main(){
    char ae=130,ao=162;
 short ocho, cinco, cuatro, tres, dos, uno;

 // 8 en binario: 0000 0000 0000 1000
 ocho = 8;
 // 5 en binario: 0000 0000 0000 0101
 cinco = 5;
 // 4 en binario: 0000 0000 0000 0100
 cuatro = 4;
 // 3 en binario: 0000 0000 0000 0011
 tres = 3;
 // 2 en binario: 0000 0000 0000 0010
 dos = 2;
 // 1 en binario: 0000 0000 0000 0001
 uno = 1;
 printf("Operadores aritm%cticos\n",ae);
double res = (double)cinco/dos; // Cast
 printf("5 / 2 = %lf\n",res);
printf("5 modulo 2 = %d\n",cinco%dos);
 printf("Operadores l%cgicos\n",ao);
 printf("8 >> 2 = %d\n",ocho>>dos);
 printf("8 << 1 = %d\n",ocho<<1);
 printf("5 & 4 = %d\n",cinco&cuatro);
 printf("3 | 2 = %d\n",tres|dos);

 printf("\n");
 return 0;
}
```

```
C:\Users\twins\Desktop\Lenguaje C\Practica7>codigoOperadores.exe
Operadores aritméticos
5 / 2 = 2.500000
5 modulo 2 = 1
Operadores lógicos
8 >> 2 = 2
8 << 1 = 16
5 & 4 = 4
3 | 2 = 3


C:\Users\twins\Desktop\Lenguaje C\Practica7>
```

areaPerimetro.c | almacenarV.c | estYCons.c | codigoOperadores.c | expLog.c

```c
#include <stdio.h>
/*
 Este programa ejemplifica el manejo de operaciones relacionales y los
 operadores lógicos. También muestra la manera de incrementar o decrementa
 una variable y la diferencia entre hacerlo antes (pre) o después (pos).
*/
int main (){
 int num1, num2, res;
 char c1, c2;
 char in=168, ao=162;
 double equis = 5.5;
 num1 = 7;
 num2 = 15;
c1 = 'h';
 c2 = 'H';

 printf("Expresiones de relaci%cn\n",ao);
 printf("%c num1 es menor a num2 ? -> \t%d\n",in,num1<num2);
 printf("%c c1 es igual a c2 ? -> \t%d\n",in,c1==c2);
 printf("%c c1 es diferente a c2 ? -> \t%d\n",in,c1!=c2);

 printf("\nExpresiones l%cgicas\n",ao);
 printf("%c num1 es menor a num2 Y c1 es igual a 'h' ? -> \t%d\n",
 in,num1<num2 && c1 == 'h');
 printf("%c c1 es igual a 's' O c2 es igual a 'H' ? -> \t%d\n",
 in,c1 == 's' || c2 == 'H');
 printf("%c c1 es igual a 's' O c2 es igual a 'S' ? -> \t%d\n",
 in,c1 == 's' || c2 == 'S');

 printf("\nIncrementos y decrementos\n");
 printf("num1++ -> \t%d\n",num1++);
 printf("--num1 -> \t%d\n",--num1);
 printf("++equis -> \t%lf\n",++equis);

 return 0;
}
```

```
C:\Users\twins\Desktop\Lenguaje C\Practica7>expLog.exe
Expresiones de relación
¿ num1 es menor a num2 ? ->     1
¿ c1 es igual a c2 ? ->         0
¿ c1 es diferente a c2 ? ->     1

Expresiones lógicas
¿ num1 es menor a num2 Y c1 es igual a 'h' ? ->         1
¿ c1 es igual a 's' O c2 es igual a 'H' ? ->    1
¿ c1 es igual a 's' O c2 es igual a 'S' ? ->    0

Incrementos y decrementos
num1++ ->       7
--num1 ->       7
++equis ->      6.500000
```

Conclusión

Con la ayuda de la practica conocemos las diferentes variables que podemos utilizar de acuerdo con la situación que se nos presente, también comprendemos como lograr que en nuestro código la variable sea declarada apropiadamente para su modificación. También aprendemos como una variable va cambiando durante nuestro proceso y como es que las variables se “imprimen” para que el usuario pueda usarla en la situación dada.

Referencias

- Nakayama Cervantes, A., Castañeda Perdomo, M., Solano Gálvez, J. A., García Cano, E. E., Sandoval Montaño, L., & Arteaga Ricci, T. I. (2018, 6 abril). *Manual de prácticas del laboratorio de Fundamentos de programación*. Laboratorio de Computación Salas A y B. http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf