



Universidad Nacional Autónoma de México
Facultad de Ingeniería
Computación para Ingenieros
Tema 6
DISEÑO DE PROGRAMAS PARA LA RESOLUCIÓN
DE PROBLEMAS DE INGENIERÍA

6. Diseño de programas para la resolución de problemas de ingeniería

Objetivo: Aplicar el método de Diseño de Programas en la elaboración de programas que resuelvan problemas básicos de ingeniería.

6. Diseño de programas para la resolución de problemas de ingeniería

6.1 Teoría del diseño de programas.

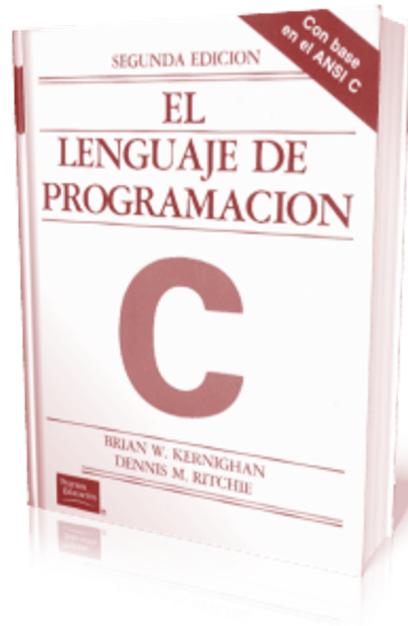
6.2 Vinculación del diseño de programas al conocimiento algorítmico.

6.3 Características básicas de un programa en lenguaje C.

6.4 Elementos y estructuras del lenguaje C en el diseño de programas.

6.5 Elaboración de programas básicos de ingeniería.

Bibliografía



El lenguaje de programación C.
Brian W. Kernighan, Dennis M. Ritchie,
segunda edición, USA, Pearson Educación 1991.



6.1 Teoría del diseño de programas.

6.1 Teoría del diseño de programas.

Un programa es un conjunto de líneas de código escritas en un lenguaje de programación determinado.

Un código escrito en un lenguaje de alto nivel no puede ser entendido por la computadora, por ello es necesario traducirlo a código máquina.

El proceso para convertir un lenguaje de alto nivel a código máquina está compuesto por dos partes principales: la compilación y el enlazado.

En la fase de compilación el compilador traduce cada uno de las partes del programa y crea módulos objeto (obj). En la fase de enlazado se unen los módulos obj, creando el módulo ejecutable.

Metodo de diseño de programas

Para desarrollar un proyecto de software es necesario establecer un enfoque disciplinado y sistemático.

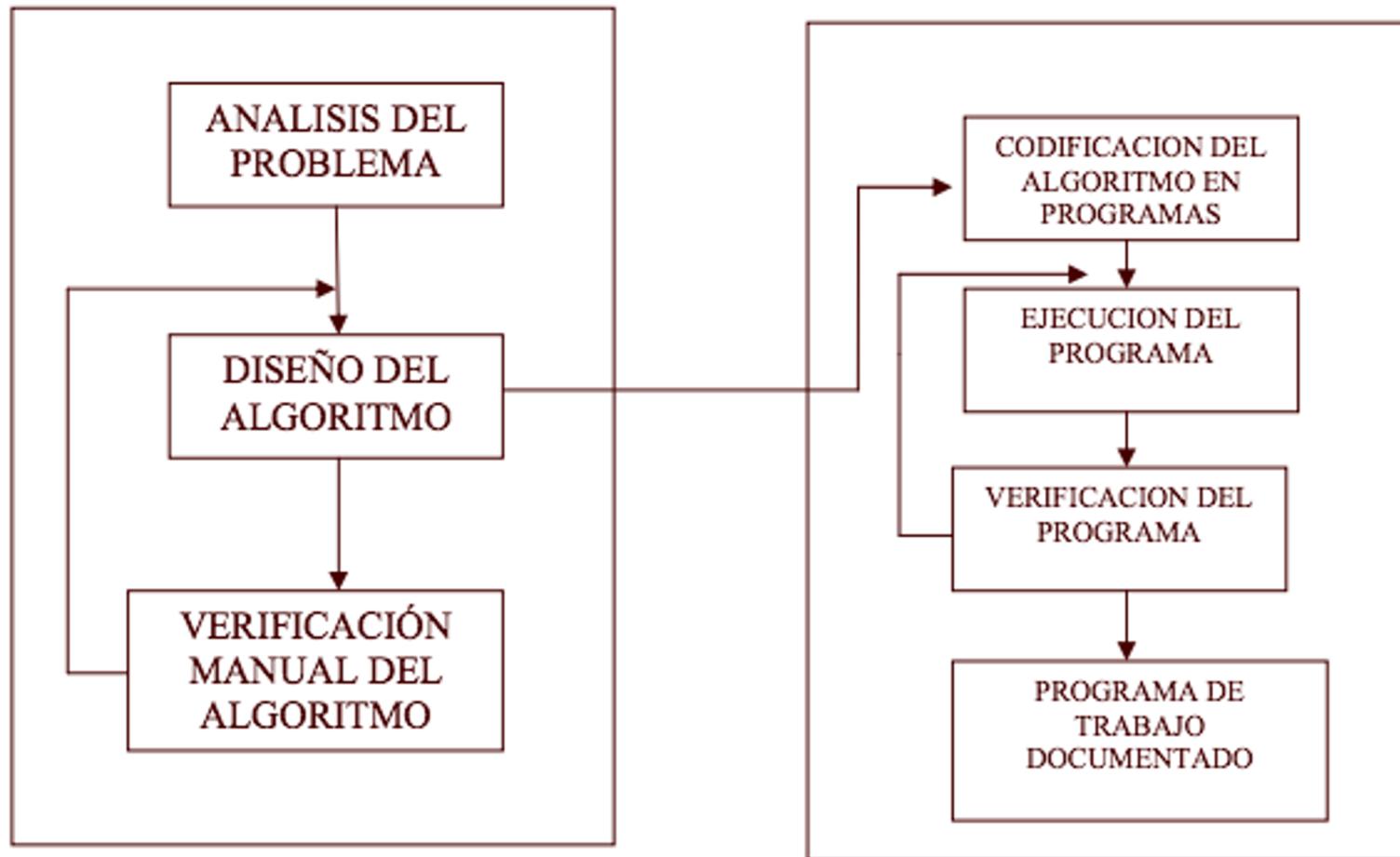
Las metodologías de desarrollo influyen directamente en el proceso de construcción de un programa y se elaboran a partir del marco definido.

La metodología para el diseño de programas se considera como un conjunto de pasos y procedimientos que se deben seguir para desarrollar software de calidad.

Por tanto, el diseño de programas conlleva el uso de un algoritmo que defina los pasos que se deben seguir para solucionar un problema (diagrama de flujo o pseudocódigo).



Diseño de una solución completa

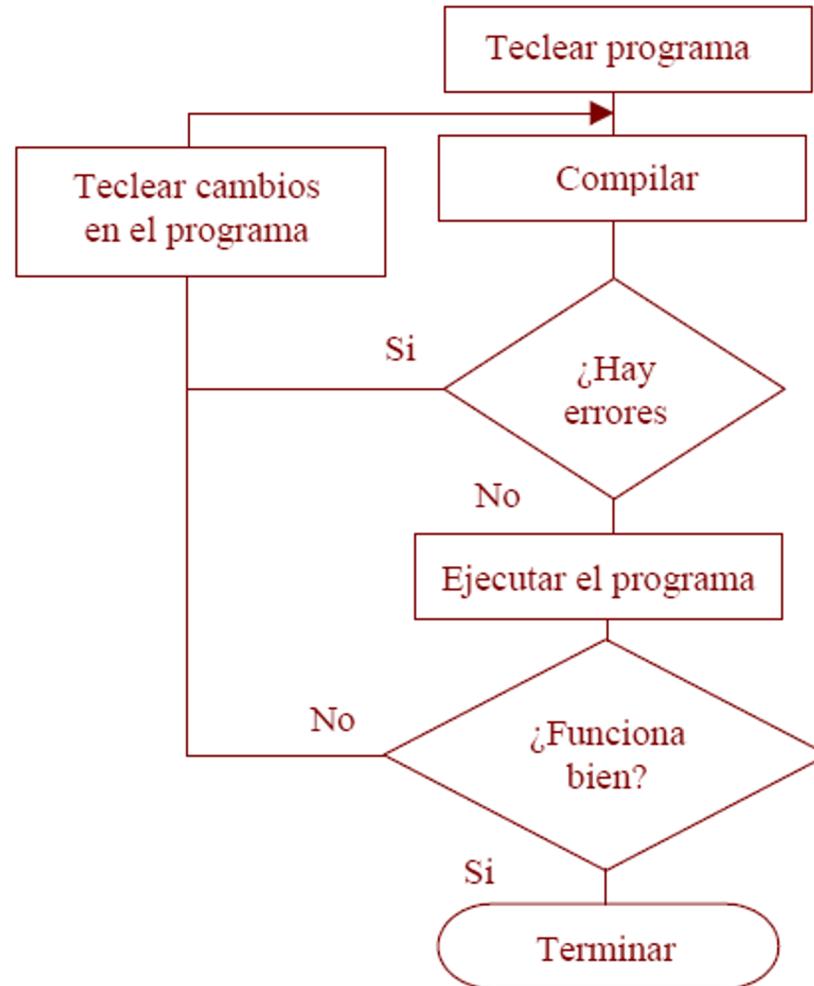


Proceso de creación de un programa en C

Para crear un programa se deben seguir las siguientes tres etapas:

- **Edición:** Utilizar un editor para escribir el programa fuente.
- **Compilación:** Compilar el programa fuente, es decir, traducir el programa a lenguaje máquina (crear el archivo obj).
- **Ejecución:** Una vez que se ha generado el archivo objeto en lenguaje máquina se ejecuta el programa.

Proceso para crear un programa



Proceso de compilación de un programa en C

Las etapas por las que pasa la compilación de un programa en C son:

- Preprocesador: se encarga de retirar los comentarios del código fuente e interpreta las directivas del preprocesador. Las directivas del procesador están compuestas por las líneas que inician con el símbolo #.

- **Compilador:** se encarga de traducir el código fuente (**texto plano**) en código ensamblador (**mnemónicos**). El código fuente con el que trabaja el compilador es el que recibe del preprocesador.
- **Ensamblador:** crea el código ejecutable (**código objeto**) a partir del código ensamblador que recibe del compilador. Normalmente los archivos ejecutables poseen la terminación **.o** y **.out**.

- **Ligador:** el ligador se encarga de combinar las funciones de alguna biblioteca o archivo especificado. En esta etapa, las referencias a variables externas son resueltas.

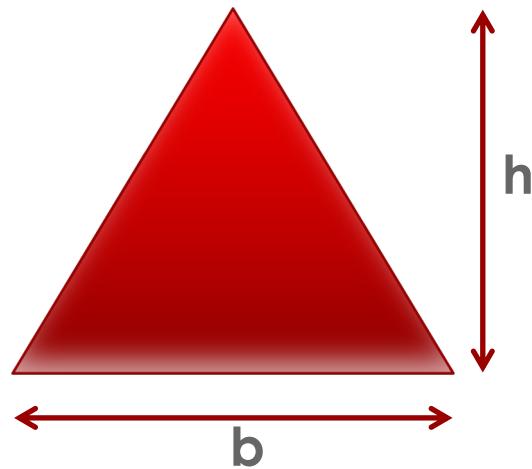
Modelo de compilación de C



Ejemplo 6.1

```
000101110100010001111111110100000  
010010110000110101101101011011001000  
1011000001010110010001000011100010011  
100110010110100110100111101110111  
0110100#include <stdio.h>  
00100110  
0001001int main()  
0101001{  
11001100    printf("Hello World")  
100000111    return 42;  
011010001000111000110001101000110  
001001101111010111011100000010100011  
00010010001011001001110111010001011  
101001110011010111000101010100010001  
0001100000110111110101001111110001
```

Crear un programa en lenguaje C que permita calcular el área de un triángulo utilizando la metodología para el diseño de programas.



Análisis del problema

La fórmula para obtener el área de un triángulo es la siguiente:

$$A = (b * h) / 2$$

De la fórmula anterior se entiende que la base del triángulo así como su altura son variables y pueden ser valores enteros o fraccionarios.

Análisis del problema (cont)

Valores de entrada:

- b: base del triángulo (puede ser entera o fraccionaria).
- h: altura del triángulo (puede ser entera o fraccionaria).

Valores de salida:

- a: área del triángulo obtenida de la fórmula $A=(b*h)/2$. El área puede ser un valor entero o fraccionario.

Ejemplo 6.2

Diseño de la solución

A partir de los valores de entrada b (base) y h (altura) que son dos números de tipo flotante, se obtiene el valor de a (área) a partir de la siguiente fórmula:

$$A = (b * h) / 2$$

El resultado de la operación se almacena en una variable de tipo flotante. La variable a posee el valor buscado.

Codificación de la solución

A partir del diseño de la solución es posible crear el programa.

```
#include <stdio.h>

int main () {
    float b, h, a;
    b = 2;
    h = 4;
    a = (b*h)/2;
    return 75;
}
```

Por lo tanto, la metodología para el diseño de programas dicta que para realizar un programa de software se deben generar una serie de pasos ordenados que lleven a la solución del problema.

Ésta es la definición de algoritmo y, por tanto, es posible elaborar un programa a partir de un algoritmo en cualquier representación (gráfico o escrito).



6.2 Vinculación del diseño de programas al conocimiento algorítmico.

6.2 Vinculación del diseño de programas al conocimiento algorítmico.

Una vez que un algoritmo está diseñado, representado y verificado se debe pasar a la etapa de implementación (codificación).

La programación (o implementación) de un algoritmo conlleva tres etapas básicas: codificación, ejecución y comprobación.

Si un algoritmo está bien realizado y representado, la codificación es transparente siguiendo la representación dada.

Es posible construir programas a partir tanto de pseudocódigo como de diagramas de flujo.

Ejemplo 6.3

Realizar la suma de dos números reales. Los números deben ser proporcionados por el usuario.

$$a + b$$

Ejemplo 6.3

A partir del problema anterior se generó el siguiente pseudocódigo que lo solventa:

INICIO

```
uno, dos : REAL
ESCRIBIR "Programa que suma dos números reales"
ESCRIBIR "Ingrese el primer número"
LEER uno
ESCRIBIR "Ingrese el segundo número"
LEER dos
res : REAL
res := uno + dos
ESCRIBIR "El resultado es " res
```

FIN

Ejemplo 6.3

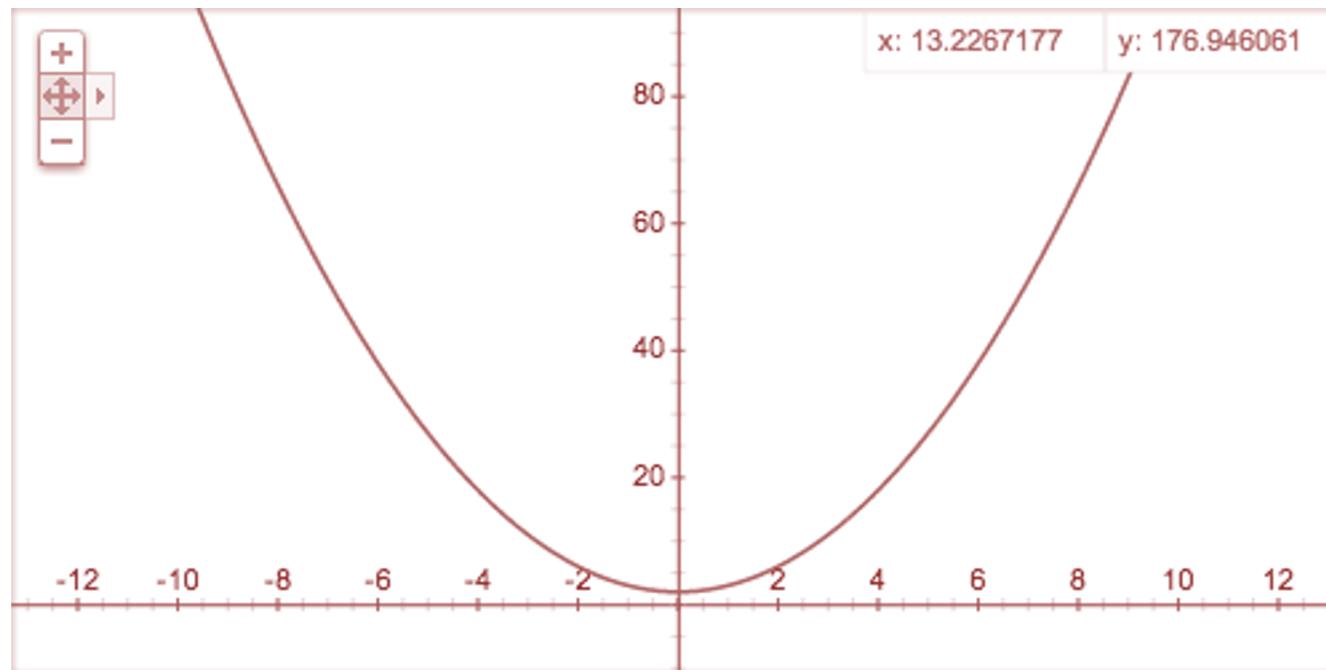
A partir del pseudocódigo que resuelve el problema es posible crear un programa de manera casi inmediata:

```
#include <stdio.h>

main () {
    float uno, dos;
    printf ("Programa que suma dos números reales");
    printf ("Ingrese el primer número");
    scanf ("%f",&uno);
    printf ("Ingrese el segundo número");
    scanf ("%f",&dos);
    float res;
    res = uno + dos;
    printf ("El resultado es %f", res);
}
```

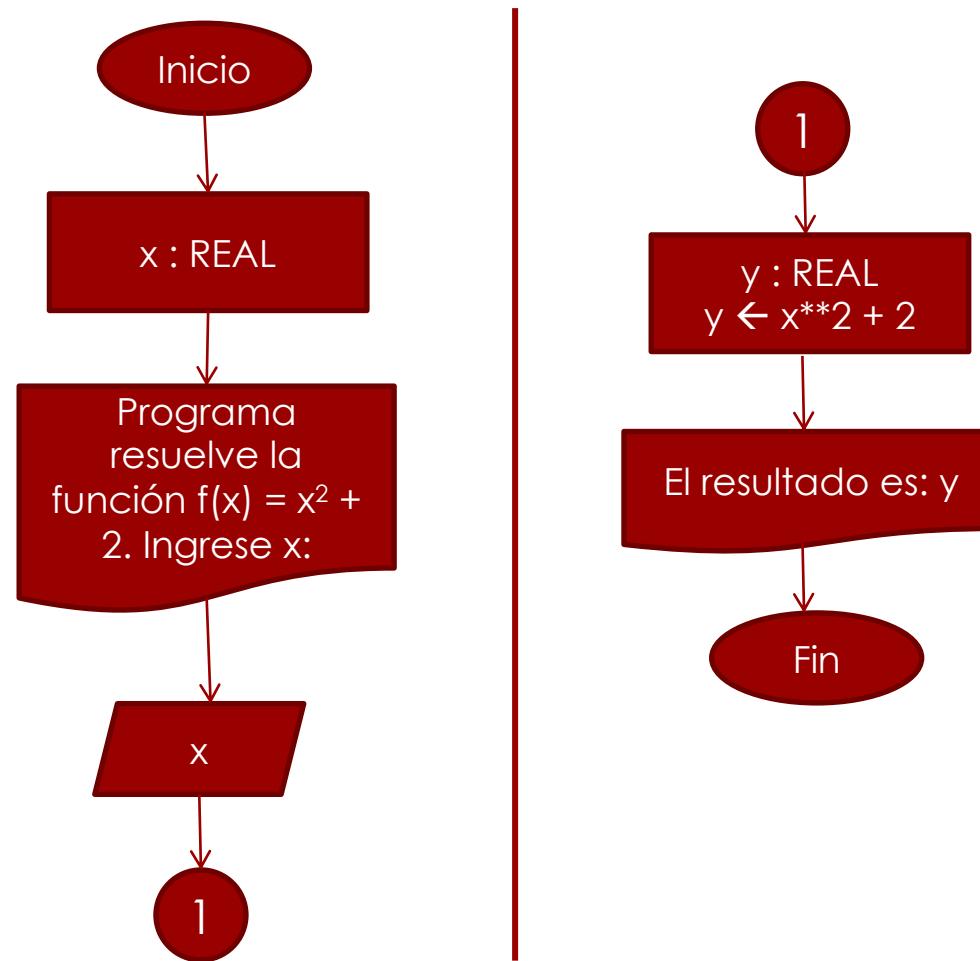
Ejemplo 6.4

Obtener el valor de la función $f(x)$ para una 'x' dada por el usuario. $f(x) = x^2 + 2$



Ejemplo 6.4

Para resolver el problema dado se generó el siguiente diagrama de flujo:



Ejemplo 6.4

A partir del diagrama de flujo anterior se puede obtener un código de programa:

```
#include <stdio.h>

main () {
    float x;
    printf ("Programa resuelve la función f(x) = x2 + 2.");
    printf ("Ingrese el valor de x: ");
    scanf ("%f",&x);
    float y;
    y = x*x + 2;
    printf ("La función valuada en %f es %f", x, y);
}
```



6.3 Características básicas de un programa en lenguaje C.

6.3 Características básicas de un programa en lenguaje C.

El proceso de desarrollo del lenguaje C se origina con la creación de un lenguaje llamado BCPL, que fue desarrollado por Martin Richards.

BCPL tuvo influencia en un lenguaje llamado B, el cual se uso en 1970 y fue inventado por Ken Thompson y que permitió el desarrollo de C en 1971, el cual lo inventó e implementó Dennis Ritchie.

C es que es un lenguaje de programación estructurado, lo que permite generar código claro y sencillo, debido a que esta basado en la modularidad.

El lenguaje de programación C, esta compuesto de tres partes fundamentales: un preprocesador, un compilador y un conjunto de librerías.

C es un lenguaje de propósito general basado en el paradigma estructurado.

El teorema del programa estructurado, demostrado por Böhm-Jacopini, dicta que todo programa puede desarrollarse utilizando únicamente 3 instrucciones de control:

- Secuencia
- Selección
- Iteración.

- **Secuencial:** Una instrucción no se ejecuta hasta que termina la anterior.
- **Selección:** Permite que el programa se bifurque en una u otra instrucción según se cumpla o no una condición lógica.
- **Iteración:** Bucles o ciclos de control que ejecutan una secuencia de instrucciones mientras se cumple la condición establecida.

Estructura de un programa en lenguaje C

Un programa en C consiste en una o más funciones (divide y vencerás), de las cuales una de ellas debe llamarse `main()` y es la principal.

Al momento de ejecutar un programa objeto (código binario), el compilador ejecutará únicamente las instrucciones que estén definidas dentro de la función principal.

Los pasos básicos para crear un programa en lenguaje C son:

- **Definir las bibliotecas a utilizar.** Las bibliotecas son archivos donde están definidas funciones útiles para el lenguaje.
- **Definir, por lo menos, una función, la función principal (main).**

La firma o sintaxis de una función tiene la siguiente forma:

```
tipoValorRetorno nombreFunción ()  
{  
    variables;  
    sentencias;  
    funciones;  
    instrucción;  
    return tipoValorRetorno      // devuelve valor  
}  
                                // Inicio de la función.  
                                }  
                                Cuerpo de la función.  
                                // Fin de la función.
```

El tipo ValorRetorno se refiere al tipo de dato que la función va a devolver. Existen diferentes tipos de datos que se analizarán más adelante.

El nombre de la función puede ser alfanumérico, debe iniciar con una letra y puede contener guión bajo.

La función está delimitada por las llaves { y }, cualquier actividad que se desee ejecutar cuando se llame esa función se debe ingresar entre esas llaves.

Tipos y tamaños 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.
- **Dobles:** números reales de doble precisión.

Variables de tipo entero en lenguaje C

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	32	-2 147 483 648	2 147 483 647
unsigned long	32	0	4 294 967 295
enum	16	-32 768	32 767

Si se omite el clasificador, por defecto se considera “signed”.

Variables de tipo punto flotante en lenguaje C:

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

Las variables de tipo flotante siempre poseen signo.

Cada tipo de dato posee un especificador para el lenguaje:

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

➤ Identificador

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

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

En C un identificador sigue las reglas antes mencionadas. A continuación se muestran algunos ejemplos de identificadores:

var1
bin2hex
otro_numero
MAX_LONG

➤ Constante

Un tipo de dato constante no cambia su valor durante la ejecución del programa. Una constante en C se puede declarar de la siguiente manera:

const int a = 1;

const double e = 2.71828182845905;

Existe otro tipo de dato constante conocido como enumeración. Una variable enumerador se puede crear de la siguiente manera:

```
enum boolean {NO, YES};
```

La primera cadena de la enumeración (NO) posee el valor 0, la siguiente cadena (YES) posee el valor 1 y, si hubiese más cadenas, así sucesivamente. Los valores por defecto dentro de una enumeración son constantes.

➤ **Variable**

Un tipo de dato **variable** puede cambiar su valor durante la ejecución del programa. Las variables pueden ser de cualquier tipo de dato simple (carácter, entero o real) e incluso arreglos.

En C una variable tiene la siguiente sintaxis:

tipoDeDato nombreVariable

Algunos ejemplos de declaración de variables de distintos tipos son:

```
float a1, b1;    //variables de tipo flotante  
int edad;        // variable de tipo entero  
char letra;      // variable de tipo carácter
```

➤ Asignación

Para determinar el valor de una variable se utiliza el símbolo de asignación. La asignación es una operación destructiva debido a que el valor que tiene una variable es borrado al momento de realizar una asignación, adquiriendo el nuevo valor dado.

En C la asignación se realiza mediante el símbolo `=`. Por ejemplo:

```
int a = 5;
```

➤ Operadores aritméticos

Las siguiente tabla muestra las operaciones aritméticas básicas en lenguaje C.

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

➤ Operadores lógicos

Las siguiente tabla muestra las operadores lógicos a nivel de bits en C.

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

➤ Expresiones lógicas

Las expresiones lógicas están constituidas por números, caracteres, constantes o variables que están relacionados entre sí por operadores lógicos. 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	$\neg p$
$\&\&$	Y	$a > 0 \&\& a < 11$
$\ $	O	$opc == 1 \ salir != 0$

➤ Ámbito o alcance de las variables

Las variables tienen un tiempo de vida que depende de la posición donde se declaren. En C existen dos tipos de variables con base en su alcance: variables locales y variables globales.

Como ya se mencionó, un programa en C puede contener varias funciones. Las variables que se declaran dentro de cada función se conocen como variables locales (a la función). Existen al momento de que la función es llamada y desaparecen cuando la función llega a su fin.

```
void sumar() {  
    int x;  
        // ámbito de la variable x  
}
```

Las variables que se declaran fuera de cualquier función se llaman variables globales. Las variables globales existen durante la ejecución de todo el programa y pueden ser utilizadas por cualquier función.

```
#include <stdio.h>

int resultado;

void multiplicar() {
    resultado = 5 * 4;
}
```

➤ Comentarios

Cuando el código de un programa es muy grande, ayuda mucho para la fase de depuración comentar bloques de código funciones para dar idea de la actividad que realiza dicho bloque del programa.

C maneja dos tipos de comentarios: **comentarios simples** y **comentarios por bloque**.

Los comentarios simples o por línea: al inicio de la línea se agregan los símbolos `//`.

Los comentarios por bloque: se pueden comentar varias líneas a la vez encerrando el bloque de código entre los símbolos `/*` y `*/`.

Los comentarios no son tomados en cuenta por el compilador debido a que el preprocesador los elimina del código fuente.

➤ Incrementos y decrementos

C maneja operadores para manejar incrementos y decrementos de un número.

El operador `++` agrega 1 a su operando. Es posible manejar preincrementos (`++n`) y posincrementos (`n++`).

El operador `--` resta 1 a su operando. Se pueden manejar predecrementos (`--n`) y posdecrementos (`n--`).

➤ Secuencias de escape

Las secuencias de escape están constituidas por dos caracteres aunque representan uno solo. A continuación se listan:

\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

Ejemplo 6.5

```
#include <stdio.h>

main () {
    double a = 5.205;
    printf("%f\n",a);
    printf("%lf\n",a);
    printf("%g\n",a);
    printf("%e\n",a);
    int b = 10;
    printf("%o\n",b);
    printf("%x\n",b);
    printf("%i\n",b);
    printf("%d\n",b);
    char c = 'a';
    printf("%c\n",c);
    printf("%d\n",c);
    printf("%i\n",c);
    printf("%o\n",c);
}
```



6.4 Elementos y estructuras del lenguaje C en el diseño de programas.

6.4 Elementos y estructuras del lenguaje C en el diseño de programas.

Las estructuras de control de flujo en un lenguaje especifican el orden en que se realiza el procesamiento de datos.

Estructuras de selección

Las estructuras de selección permiten realizar una u otra acción con base en una condición lógica. Es importante aclarar que las opciones a realizar son mutuamente excluyentes.

Estructura de selección if

La estructura de selección más conocida es la estructura if. Su sintaxis es la siguiente:

```
if (condición){  
    /* Código a ejecutar si la condición  
       valuada es verdadera */  
}
```

Ejemplo 6.6

```
#include <stdio.h>

void main () {
    int a;
    printf ("Ingresar número");
    scanf ("%d", &a);
    if (a < 0 ) {
        a = a*-1;
    }
    printf ("El valor absoluto de a es: %d", a);
}
```

Estructura de selección if-else

La estructura de selección if completa involucra una segunda parte conocida como else. Su sintaxis es la siguiente:

```
if (condición){  
    /* Código a ejecutar si la condición  
       valuada es verdadera */  
} else {  
    /* Código a ejecutar si la condición  
       valuada es falsa */  
}
```

Ejemplo 6.7

```
#include <stdio.h>

int main () {
    int a;
    printf ("Ingresar número");
    scanf ("%d", &a);
    if (a%2 == 0 ) {
        printf ("El número %d es par.", a);
    } else {
        printf ("El número %d es impar.", a);
    }
    return 48;
}
```

Pueden existir tantas estructuras if-else anidadas como se requieran:

```
if (condición) {  
    if (condición)  
        if (condición)  
            //      Instrucciones  
        else  
            //      Instrucciones  
    else  
        //      Instrucciones  
} else {  
    if (condición)  
        //      Instrucciones  
    else  
        if (condición)  
            //      Instrucciones  
        else  
            //      Instrucciones  
}
```

Estructura de selección switch-case

La estructura switch-case permite valuar una variable entre varias opciones. Su sintaxis es la siguiente:

```
switch (opcion){  
    case valor1:  
        /* Código a ejecutar*/  
        break;  
    case valor2:  
        /* Código a ejecutar*/  
        break;  
    ...  
    case valorN:  
        /* Código a ejecutar*/  
        break;  
    default:  
        /* Código a ejecutar*/  
}
```

Ejemplo 6.8

```
#include <stdio.h>

void main () {
    int mes;
    printf ("Proporcione el mes de su cumpleaños: ");
    scanf ("%d", &mes);
    switch (mes) {
        case 1:
            printf("Usted nació en Enero.");
            break;
        case 2:
            printf("Usted nació en Febrero.");
            break;
        case 3:
            printf("Usted nació en Marzo.");
            break;
```

Ejemplo 6.8

```
case 4:  
    printf("Usted nació en Abril.");  
    break;  
case 5:  
    printf("Usted nació en Mayo.");  
    break;  
case 6:  
    printf("Usted nació en Junio.");  
    break;  
case 7:  
    printf("Usted nació en Julio.");  
    break;  
case 8:  
    printf("Usted nació en Agosto.");  
    break;  
case 9:  
    printf("Usted nació en Septiembre.");  
    break;
```

```
case 10:  
    printf("Usted nació en Octubre.");  
    break;  
case 11:  
    printf("Usted nació en Noviembre.");  
    break;  
case 12:  
    printf("Usted nació en Diciembre.");  
    break;  
default:  
    printf ("Opción inválida.");  
}  
}
```

Estructura de selección condicional

La expresión condicional (también llamado operador ternario) permite realizar una comparación rápida. Su sintaxis es la siguiente:

Condición ? SiSeCumple : SiNoSeCumple

Se valúa una condición, si la condición se cumple (verdadera) se ejecuta la instrucción después del símbolo ?; si la condición no se cumple (falso), se ejecuta la instrucción después de ::.

Ejemplo 6.9

```
#include <stdio.h>

int main () {
    int a, b;
    printf ("\nPrograma que calcula el error matemático.");
    printf ("\nentre dos números |a - b|.");
    printf ("\nProporcione a: ");
    scanf ("%d", &a);
    printf ("\nProporcione b: ");
    scanf ("%d", &b);
    int c = a<b ? b-a : a-b;
    printf ("\nE = %d\n",c);
}
```

Estructuras de repetición

Las estructuras de repetición son los llamadas estructuras cíclicas o de bucles. Permiten ejecutar un conjunto de instrucciones mientras se cumpla una condición. Existen tres estructuras de repetición: while, do-while y for.

Estructura de repetición while

La estructura del ciclo while valida la condición dada y si es correcta ejecuta el código que se encuentra dentro de las llaves, si no es correcta sigue el flujo normal del programa. Su sintaxis es la siguiente:

```
while (condición){      // Inicio del ciclo while
    /* Código a ejecutar */
}
```

// fin del ciclo while

Ejemplo 6.10

```
#include <stdio.h>

int main () {
    int a;
    a = 0;
    while (a < 10 ) {
        printf ("\v%d\t\n",a++);
        sleep(0.5);
    }
}
```

Estructura de repetición do-while

La estructura cíclica do-while ejecuta, por lo menos, el bloque de código que se encuentra dentro de las llaves y después valida la condición, si ésta se cumple vuelve a ejecutar el bloque, de lo contrario sigue con el flujo normal del programa. Su sintaxis es la siguiente:

```
do {  
    /*  
     Código a ejecutar  
    */  
} while (condición);
```

Ejemplo 6.11

```
#include <stdio.h>

int main () {
    char a = ' ';
    do {
        a = getchar();
    } while (a != 'x');
}
```

Ejemplo 6.12

```
#include <stdio.h>

int main () {
    float a, b;
    char salir = 'n';
    do {
        printf ("\nError matemAtico\n");
        printf ("\nValor real: ");
        scanf ("%f", &a);
        printf ("\nValor aproximado: ");
        scanf ("%f", &b);
        float c = a<b ? b-a : a-b;
        printf ("E = %f",c);
        printf ("\nDesea calcular otro error (s/n)");
        setbuf(stdin,NULL);
        scanf ("%c",&salir);
    } while (salir == 's');
}
```

Ejemplo 6.12

```
#include <stdio.h>
#define p printf
#define s scanf
int main () {
    float a, b;
    char salir = 'n';
    do {
        p ("\nError matemAtico\n");
        p ("\nValor real: ");
        s ("%f", &a);
        p ("\nValor aproximado: ");
        s ("%f", &b);
        float c = a<b ? b-a : a-b;
        p ("E = %f",c);
        p ("\nDesea calcular otro error (s/n)");
        setbuf(stdin,NULL);
        s ("%c",&salir);
    } while (salir == 's');
}
```

Ejemplo 6.12 c

```
#include <stdio.h>
#define p printf
#define s scanf
int main () {
    float a, b;
    char salir = 'n';
    do {
        system ("clear");
        p ("\nError matemAtico\n");
        p ("\nValor real: ");
        s ("%f", &a);
        p ("\nValor aproximado: ");
        s ("%f", &b);
        float c = a<b ? b-a : a-b;
        p ("E = %f",c);
        p ("\nDesea calcular otro error (s/n)");
        setbuf(stdin,NULL);
        s ("%c",&salir);
    } while (salir == 's');
}
```

Estructura de repetición for

La estructura for es una estructura repetitiva que consta de tres partes: un valor inicial, una condición y una expresión final. Su sintaxis es la siguiente:

```
for (valorInicial; condición; (in/de)-creemento) {  
    /*Código a ejecutar*/  
}
```

Ejemplo 6.13

```
#include <stdio.h>

int main () {
    int grado, x, fx, coef, cont;
    printf ("\nPrograma que calcula el valor de un polinomio");
    printf ("\nProporcione el grado del polinomio: ");
    scanf ("%d", &grado);
    printf ("\nProporcione la x a valuar: ");
    scanf ("%d", &x);
    fx = 0;
    for (cont = 0 ; cont <= grado ; cont++){
        printf ("\nCoeficiente de x%d: ", cont);
        scanf ("%d", &coef);
        fx = fx + coef*pow(x,cont);
    }
    printf ("\nf(x) = %d\n",fx);
}
```

Enumeración

Una enumeración es un tipo de dato constante. Su sintaxis se muestra a continuación:

```
enum nombre {elem1, elem2, ... elemn};
```

A cada elemento de la enumeración se le asigna un valor constante.

Ejemplo 6.14

```
#include<stdio.h>

main() {
    enum dias {lunes, martes, miercoles, jueves, viernes, sabado, domingo};
    enum dias varEnum;
    varEnum = viernes;
    switch(varEnum) {
        case lunes:
        case martes:
            printf("¡Inicio de semana!\n");
            break;
        case miercoles:
            printf("Ombligo de semana.\n");
            break;
        case jueves:
        case viernes:
        case sabado:
            printf("¡Inicia el fin de semana!\n");
            break;
        case domingo:
            printf("Se acabO la semana.");
            break;
    }
}
```

Arreglos

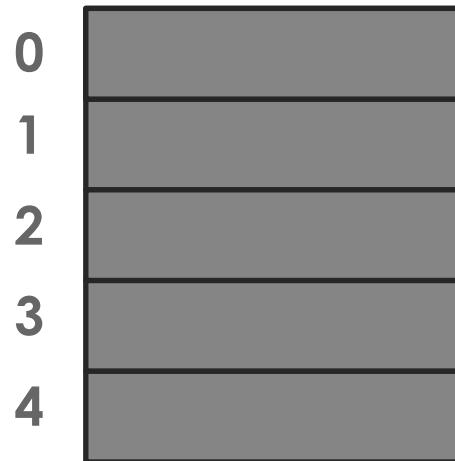
Se denomina arreglo a un conjunto de datos del mismo tipo que son contiguos y ordenados. Un arreglo posee un tamaño fijo definido al momento de crearse.

A cada localidad del arreglo (elemento) se le asocia un número. Para recorrer un arreglo es necesario utilizar un índice.

Existen arreglos unidimensionales y multidimensionales.

Arreglos unidimensionales

La primera localidad de un arreglo corresponde al índice 0 y la última corresponde al índice $n-1$, donde n es el tamaño del arreglo.



Por ejemplo, si se tiene un grupo con 5 alumnos, se pueden manejar sus calificaciones finales en un arreglo unidimensional de la siguiente manera:

Alum 0	Alum 1	Alum 2	Alum 3	Alum 4
10	8	5	8	7

La sintaxis para definir un arreglo en lenguaje C es la siguiente:

tipoDeDato nombre[tamaño]

Donde nombre se refiere al identificador del arreglo, tamaño es un número entero y define el número máximo de elementos que puede contener el arreglo. Un arreglo puede ser de cualquier tipo de dato: entero, real, carácter o estructura.

Ejemplo 6.15

```
#include <stdio.h>

#define TAMANO 5

int main () {
    int lista[TAMANO] = {10, 8, 5, 8, 7};
    int c = 0;
    while (c < 5 ) {
        printf ("\nCalif alumno %d = %d",c+1,lista[c]);
        c += 1;
    }
    printf ("\n");
    return 0;
}
```

Ejemplo 6.15 b

```
#include <stdio.h>

#define TAMANO 5

int main () {
    int lista[TAMANO] = {10, 8, 5, 8, 7};
    int c;
    for (c = 0 ; c < 5 ; c++ ) {
        printf ("\nCalif alumno %d = %d",c+1,lista[c]);
    }
    printf ("\n");
    return 0;
}
```

Ejemplo 6.16

```
#include<stdio.h>

main(){
    char palabra[20];
    int i=0;
    printf("Introduce una palabra: ");
    scanf("%s", palabra);
    printf("La palabra ingresada es: %s \n", palabra);
    for (i = 0 ; i < palabra[i] ; i++){
        printf("%c \n", palabra[i]);
    }
}
```

Arreglos multidimensionales

En C es posible crear arreglos multidimensionales almacenar mayor cantidad de información. La sintaxis para declarar arreglos multidimensionales es:

tipoDato nombre [tamaño][tamaño]...[tamaño];

Ejemplo 6.17

```
#include<stdio.h>

int main(){
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, j;
    printf("Imprimir Matriz\n");
    for (i=0 ; i<3 ; i++){
        for (j=0 ; j<3 ; j++){
            printf("%d, ",matriz[i][j]);
        }
        printf("\n");
    }
    return 42;
}
```

Funciones

En C es posible tener dentro de un archivo fuente varias funciones, esto con el fin de dividir las tareas y que sea más fácil la depuración o mejora del código.

En lenguaje C la función principal se llama main. El compilador solo ejecuta las instrucciones que se encuentran dentro de la función main, empero, ésta puede a su vez llamar o ejecutar otras funciones.

Ejemplo 6.18

```
#include<stdio.h>

void imp_rev(char s[]) {
    int t;
    for( t=strlen(s)-1; t>=0; t-- )
        printf("%c",s[t]);
    printf("\n");
}

main() {
    char nombre[]="Facultad";
    imp_rev(nombre);
}
```

Archivos

Un archivo es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros que son del mismo tipo.

Se puede conseguir la entrada y la salida de datos a un archivo a través del uso de la biblioteca de funciones de la librería STDIO.H

Apuntador a archivo

El apuntador a un archivo es el hilo común que unifica el sistema de E/S con un buffer.

Un apuntador a archivo señala a la información que contiene y define ciertas características sobre él, incluyendo el nombre, el estado y la posición actual del archivo.

Un apuntador identifica un archivo específico y utiliza la secuencia asociada para dirigir el funcionamiento de las funciones de E/S.

Un apuntador a un archivo es una variable de tipo puntero al tipo FILE que se define en STDIO.H.

La sintaxis para obtener una variable de tipo archivo es la siguiente:

FILE *F;

Abrir archivo

La función `fopen()` abre un flujo de datos para que pueda ser utilizado y lo asocia a un archivo. Su estructura es la siguiente:

```
*FILE fopen(char *nombre_archivo, char *modo);
```

Donde `nombre_archivo` es un apuntador a una cadena de caracteres que representan el nombre del archivo y puede incluir una especificación del directorio que lo contiene. La cadena a la que apunta `modo` determina como se abre el archivo.

Modo de apertura de archivo

r: Abre un archivo de texto para lectura.

w: Crea un archivo de texto para escritura.

a: Abre un archivo de texto para añadir.

r+: Abre un archivo de texto para lectura / escritura.

w+: Crea un archivo de texto para lectura / escritura.

a+: Añade o crea un archivo de texto para lectura / escritura.

Cerrar archivo

La función `fclose()` cierra una secuencia que fue abierta mediante una llamada a `fopen()`. Escribe la información que se encuentre en el buffer al disco y realiza un cierre formal del archivo a nivel del sistema operativo.

Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa.

La estructura de esta función es:

```
int fclose(FILE *apArch);
```

Donde apArch es el apuntador al archivo devuelto por la llamada a fopen(). Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

Ejemplo 6.19**fopen y fclose**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    archivo = fopen("archivo.txt", "r");
    if (archivo != NULL) {
        printf("El archivo se abrió correctamente.");
        fclose(archivo);
    } else {
        printf("Error al abrir el archivo.");
    }
    return 30;
}
```

Leer y escribir en archivo

Las funciones fgets() y fputs() pueden leer y escribir cadenas sobre los archivos. Las estructuras de estas funciones son:

```
char *fputs(char *buffer, FILE *apArch);  
char *fgets(char *buffer, int tamaño, FILE *apArch);
```

La función puts() escribe la cadena a un archivo específico. La función fgets() lee una cadena desde el archivo especificado hasta que lee un carácter de nueva línea o longitud-1 caracteres.

Ejemplo 6.20**fgets**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char caracteres [50];
    archivo = fopen("archivo.txt", "r");
    if (archivo != NULL) {
        printf("El archivo se abrió correctamente.");
        printf("Contenido del archivo\n");
        while (feof(archivo) == 0) {
            fgets (caracteres, 50, archivo);
            printf("%s", caracteres);
        }
        fclose(archivo);
    }
    return 32;
}
```

Ejemplo 6.21**fputs**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char escribir [] = “Escribir cadena en archivo mediante
                        fputs. \nCPI.”;
    archivo = fopen(“archivo.txt”, “r+”);
    if (archivo != NULL) {
        printf(“El archivo se abrió correctamente.”);
        fputs (escribir, archivo);
        fclose(archivo);
    }
    return 0;
}
```

Las funciones `fprintf()` y `fscanf()` se comportan exactamente como `printf()` y `scanf()`, excepto que operan sobre archivo. Sus estructuras son:

```
int fprintf(FILE *apArch, char *formato, ...);  
int fscanf(FILE *apArch, char *formato, ...);
```

Donde `apArch` es un puntero al archivo devuelto por una llamada a `fopen()`. `fprintf()` y `fscanf()` dirigen sus operaciones de E/S al archivo al que apunta `apArch`.

Ejemplo 6.22**fscanf**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char caracteres [50];
    archivo = fopen("archivo.txt", "r");
    if (archivo != NULL) {
        fscanf(archivo, "%s", caracteres);
        printf("%s", caracteres);
        fclose(archivo);
    }
    return 32;
}
```

Ejemplo 6.23**fprintf**

```
#include<stdio.h>

int main() {
    FILE *archivo;
    char escribir [] = “Escribir cadena en archivo mediante
                        fprintf. \nCPI.”;
    archivo = fopen(“archivo.txt”, “r+”);
    if (archivo != NULL) {
        fprintf(archivo, escribir);
        fprintf(archivo, “%s”, “\nPAMN”);
        fclose(archivo);
    }
    return 0;
}
```

```
    char(100[16<<
100(int(10),int(11)
=158    ?1==7?16:152
    171:0<1704
    (4*1314,
    (11*12)*
    14*2)
;while
18,*18=(100/14)
100/14,100=14**
--15)*(14+1)*
100/10,(100
main(int(1)
1<1)<<30,12
):1==1)?1
10,0):(main
1:1),
main++1,(  ++0,0
))>>6?1
++0)?12*(  1-40)+main
0:*(4*0+1)?  10*(main(
11>=(1039==7?10/9:1)
9);return(main
    16-((1<<30)),1*0=100+0
    15])
);int(1)
7441/2:-1:1-
0:0/14,13*0516<
14* (13?10/9*8:
(12/2),100
;char*18={
(14*0)?17?
,100-=14**
18+=,100*= 10
100|--17,
--9*14/(2) *15
main(int(1)
,char**0){
int 10,11=10=(
1==2??*(  0=1[0],main(64main(60,0),0+1
++56:-64+
1;if(1!=2)return(1<2?1?main(
(1a((1610)?main(46,0),1111
)))):1<62?
(12/10+64,
60,((++(*0
),0)))>11
?10++0):  (100(10,12),11
16-((1<<30)),1*0=100+0
    15])
;void
=1039,0
1/8,12n0
170:0/571:
0)40,17*13+
=14*0?17:-5*( 100 +11*11*12)
100 -(100*10*15
++:(1*15)=
1:100+=1*
=(100*10),
--11);)int
10,11=10=(
0=1[0],main(64main(60,0),0+1
1;if(1!=2)return(1<2?1?main(
(1a((1610)?main(46,0),1111
putchar(1)
1++0,0));
1++0,0);
;file(11=
*1),10<12*
*12,0)));)
```

6.5 Elaboración de programas básicos de ingeniería.

6.5 Elaboración de programas básicos de ingeniería.

Un programa es la culminación de un trabajo de ingeniería bien realizado que involucra el análisis de requerimientos y el diseño de la solución.

Para dar como terminado el proceso de ingeniería de software es necesario realizar pruebas al programa para observar como reacciona ante valores extremos o incorrectos.

Serie de ejercicios en lenguaje C

Realizar la siguiente serie de ejercicios. Se debe entregar el algoritmo de la solución (pseudocódigo para los primeros 5 ejercicios y diagrama de flujo para el resto de los ejercicios) y el programa (código fuente).

La solución de todos los ejercicios se debe imprimir en pantalla y en un archivo de texto.

Ejercicio 6.1

Obtener fecha anterior

Dada una fecha en la forma dd-mm-aaaa, obtener como resultado la fecha del día anterior.

Un año es bisiesto si es divisible entre 4, a menos que sea divisible entre 100. Sin embargo, si un año es divisible entre 100 y además es divisible entre 400, también resulta bisiesto.

Ejercicio 6.2

Serie de fibonacci

Una de las series más famosas es sin duda alguna la serie de Fibonacci cuya secuencia se muestra a continuación:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,...

Imprimir los primeros 150 términos de la Serie de Fibonacci.

Ejercicio 6.3

Contar caracteres

A partir de una cadena dada, obtener el número de vocales, consonantes, espacios y el número total de caracteres (sin espacios) de una cadena de texto dada.

Ejercicio 6.4

Conversión de base 10 a base b

Crear un programa que convierta un número en base 10 a base 2, 8 y 16. De cada número en base 2, 8 y 16 debe obtener sus complementos ar y ar-1.

Ejercicio 6.5

Obtener el valor de $f(x)$

Realizar un programa en lenguaje estructurado que permita calcular el valor de la función 'y' para un polinomio dado por el usuario. Los coeficientes del polinomio se reciben del usuario y se almacenan en un arreglo.

Ejercicio 6.6**Escribir los números en forma descendente**

Escribir un programa en lenguaje C tal que dados como datos 3 números enteros diferentes, escribir estos números en forma descendente.

Ejercicio 6.7**Recibir ene números**

Dados como datos n números enteros (desde 1 hasta n) obtener la suma de los números impares y el promedio de los números pares.

Ejercicio 6.8

Conjetura de Ulam

La conjetura de Ulam inicia con un número entero positivo. Si el número es par se divide entre 2; si el número es impar se multiplica por 3 y se aumenta 1. Se obtienen enteros positivos repitiendo el proceso hasta llegar a 1.

Programar en lenguaje C la conjetura de ULAM.

Ejercicio 6.9

Casilla electrónica

En una reciente elección donde hubo cuatro candidatos, se quiere encontrar el número de votos correspondientes a cada candidato y el porcentaje que obtuvo respecto al total de votantes. El usuario tecleará los votos de manera desorganizada. La cantidad de votos no está definida.

Realizar una casilla electrónica que registre los votos de la elección descrita anterior.

Ejercicio 6.10

Ordenar matriz

Dada una matriz ordenada de manera ascendente, generar otra matriz ordenada de manera descendente, es decir:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



16	15	14	13
12	11	10	9
8	7	6	5
4	3	2	1

La creación de la matriz se debe hacer desde otra función pasando como argumento la matriz original.

Ejercicio 6.11**Ordenar matriz v2**

Dada una matriz desordenada, generar otra matriz ordenada de manera ascendente, es decir:

4	2	3	11
21	14	17	18
9	10	15	12
1	19	8	5



1	2	3	4
5	8	9	10
11	12	14	15
17	18	19	21

La creación de la matriz se debe hacer desde otra función pasando como argumento la matriz original.

6. Diseño de programas para la resolución de problemas de Ingeniería

Objetivo: Aplicar el método de Diseño de Programas en la elaboración de programas que resuelvan problemas básicos de ingeniería.

6.1 Teoría del diseño de programas.

6.2 Vinculación del diseño de programas al conocimiento algorítmico.

6.3 Características básicas de un programa en lenguaje C.

6.4 Elementos y estructuras del lenguaje C en el diseño de programas.

6.5 Elaboración de programas básicos de ingeniería.