Ejemplo 1:
Programa que calcula el factorial de un número entero mayor a cero invocando una función de forma iterativa.

```
  /*
   * Calcula y muestra el factorial de un numero con driver incluido.
   */

  #include <stdio.h>
  int factorial(int n);

  /*
   * Calcula n! para n mayor o igual a cero
   */
  int factorial(int n)
  {
    int i,            /* variable local */
    producto = 1;
    /* Calcula el producto n*(n-1)*(n-2)*...*2*1 */
    for  (i = n;  i > 1;  --i) {
      producto *= i;
    }
    /* Devuelve el resultado de la funcion */
    return (producto);
  }

  /*
   * Muestra la llamada de una funcion desde la funcion principal
* que pasa un argumento a la funcion definida.
   */
  int main(void)
  {
    int num, fact;

    printf("Dar un entero entre 0 y 10> ");
    scanf("%d", &num);
    if (num < 0) {
      printf("El factorial de un numero negativo (%d) es indefinido\n", num);
    }
    else if (num <= 30) {
      fact = factorial(num);
      printf("El factorial de %d es %d\n", num, fact);
    }
    else {
      printf("Numero fuera de rango: %d\n", num);
    }
    return (0);
  }
```

Ejemplo 2:
Programa para el cálculo del número de combinaciones que se obtienen de seleccionar r elementos de un conjunto de n. La fórmula para éste cálculo es:

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

```
/*
   * Calcula el numero de combinaciones de n elementos
   * tomados r al tiempo.
   */
#include <stdio.h>
int factorial(int n);

  /*
   * Calcula n! para n mayor o igual a cero
   */
  int factorial(int n)
  {
    int i,            /* variable local */
```

```
  producto = 1;
  /* Calcula el producto n*(n-1)*(n-2)*...*2*1 */
  for  (i = n;  i > 1;  --i) {
    producto *= i;
  }
  /* Devuelve el resultado de la funcion */
  return (producto);
}

/*
 * Muestra multiples llamadas desde la funcion principal pasando
 * diferentes argumentos de llamada.
 */

int main(void)
{
  int n, r, c;

  printf("Dar numero total de componentes> ");
  scanf("%d", &n);
  printf("Dar numero de componentes seleccionados> ");
  scanf("%d", &r);
  if (r <= n) {
    c = factorial(n) / (factorial(r) * factorial(n-r));
    printf("El numero de combinaciones es %d\n", c);
  }
  else {
    printf("Los componentes seleccionados no pueden exceder el numero total\n");
  }
  return (0);
}
```

Ejemplo 3:

Función que devuelve el valor redondeado de su primer argumento al número de posiciones decimales indicado por su segundo argumento.

```
#include <stdio.h>
#include <math.h>

/*
 * Redondea el valor de x al numero de decimales indicado por posiciones .
 * El argumento posiciones es mayor o igual a cero.
 */

double redondea(double x, int posiciones)
{
  int    signo;        /* -1 si x es negativo, 1 en otro caso  */
  double potencia,       /* elevacion por 10 por los lugares indicados  */
         temp_x,      /* copia de |x| con puntos decimales lugares movidos
                       a la derecha. */
         redondeo_x;   /* resultado de la funcion  */

  /* Guarda el signo de x  */
  if (x < 0)
    signo = -1;
  else
    signo = 1;
  /* Calcula valor redondeado */
  if (posiciones >= 0) {
    potencia = pow(10.0, posiciones);
    temp_x = fabs(x) * potencia;
    redondeo_x = floor(temp_x + 0.5) / potencia * signo;
  }
  else {
    printf("\nError: el segundo argumento no puede ser negativo.\n");
    printf("No se redondea.\n");
    redondeo_x = x;
  }
  return (redondeo_x);
}
```

Ejemplo 4:

Función que devuelve 1 si su argumento es par o 0 si es impar.

```
/*
 * Indica si num es par (divisible por 2):
 * devuelve 1 si es par, 0 si no lo es
 */

int par(int num)
{
  int test;

  test = ((num % 2) == 0);
  return (test);
}
```

Ejemplo 5:

Programa para hallar el menor divisor de un número o determina si es un número primo.

Análisis del programa principal

Requerimientos de datos:

Constante del problema

NMAX 1000 /* mayor numero a probar */

Dato del problema

int n /* numero a comprobar si es primo */

Salida del problema

int min_div /* minimo divisor (mayor a 1) de n */

Diseño

Algoritmo inicial:

1. Lee número a comprobar si es primo
2. Halla el menor divisor diferente de 1, o determina que el número es primo
3. Imprime el menor divisor o el mensaje que el número es primo

Refinamiento del Algoritmo:

Refinamiento paso 1:

1.1. Lee número n
1.2. if n < 2
  Imprime mensaje de error
 else if n <= NMAX
  realizar pasos 2 y 3
 else
  Imprime mensaje de error

Refinamiento paso 3:

1.1. if el menor divisor es n
  Imprime mensaje que es n es primo
 else
  Imprime el menor divisor de n

Programación del programa principal

```
/*
 * Calcula y muestra el menor divisor (diferente a 1) del entero n.
 * Indica si n es primo si no se encuentra un divisor menor a n.
 */
#include <stdio.h>
```

```
#define NMAX 1000

int main(void)
{
  int n,       /* numero a comprobar si es primo */
      min_div; /* minimo divisor (mayor a 1) de n */

/* Lee el numero. */
  printf("dar un numero > ");
  scanf("%d", &n);
  /* Comprueba si el numero esta en el rango entre 2...NMAX */
  if (n < 2) {
    printf("Error: numero muy pequeno. El menor primo es 2.\n");
  }
  else if (n <= NMAX) {
  /* Halla el menor divisor (> 1) de n */
    min_div = halla_div(n);
  /* Muestra el menor divisor o un mensaje que n es primo. */
    if (min_div == n)
      printf("%d es un numero primo.\n", n);
    else
      printf("%d es el menor divisor de %d.\n", min_div, n);
  }
  else {
    printf("Error: El mayor numero aceptado es %d.\n", NMAX);
  }
  return (0);
}
```

Análisis de la función halla_div

Requerimientos de datos:

Dato de la función

int n /* numero a comprobar si es primo */

Salida de la función (a retornar)

int divisor /* minimo divisor (mayor a 1) de n */

Variables de programa

int intento /* prueba de divisor despues de 2 */

Diseño

Algoritmo inicial:

1. if n es par
   asignar divisor a 2
   else
   asignar divisor a 0 (no hay divisor)
   asignar intento a 3
2. mientras divisor sea 0, probar siguiente número impar. Si se halla un divisor almacenar en divisor. Si intento supera sqrt(n) almacena n en divisor
3. devuelve divisor

Refinamiento del paso 2:

1.1. mientras divisor sea 0
1.2. if intento > sqrt(n)
divisor = n
else if intento es divisor de n
divisor = intento
else
intento = intento + 2

Programación de la función halla_div

```
#include <math.h>
/*
 * Halla el menor divisor de n entre 2 y n (n es mayor que 1)
 */

int halla_div(int n)
{
  int intento,   /* candidato a menor divisor de n  */
      divisor; /* menor divisor de n; cero indica que no se encuentra */

  /* Inicializa divisor e intento dependiendo de
     si n es par o impar.   */
  if (par(n)) {
    divisor = 2;
  }
  else {
    divisor = 0;
    intento = 3;
  }
  /* Prueba cada numero impar como divisor de n hasta encontrar un divisor
     o hasta que intento es tan grande que n es el menor divisor. */
  while (divisor == 0) {
    if (intento > sqrt(n))
      divisor = n;
    else if ((n % intento) == 0)
      divisor = intento;
    else
      intento += 2;
  }
 /* Devuelve el menor divisor de n. */
  return (divisor);
}
```

Ejemplo 6:

Función que utiliza la función anterior para hallar los factores primos de un número n (n>1).

```
/*
 * Muestra los factores primos de n (n > 1).
 * Ejemplo:  factor(12) imprime  12 = 2 x 2 x 3
 */

void factor(int n)
{
  int factor_rem,    /* producto de factores remanentes de n */
      factor_curso;   /* factor en curso de n */

/* Muestra parte inicial del mensaje incluyendo el menor factor primo */
  factor_curso = halla_div(n);
  printf("%d = %d", n, factor_curso);

  /* Halla y muestra los factores restantes precedidos por el signo x. */
  for  (factor_rem = n / factor_curso;
        factor_rem > 1;
        factor_rem /= factor_curso) {
    factor_curso = halla_div(factor_rem);
    printf(" x %d", factor_curso);
  }
  printf("\n");
}
```

Ejemplo 7:

Programa que utiliza una función que separa un número real en tres partes: signo, parte entera y parte fraccionaria. Muestra el uso de paso de argumentos por referencia.

```
/*
 * Muestra el uso de una funcion con parametros de entrada y salida.
 */
```

```
#include <stdio.h>
#include <math.h>

/*
 * Separa un numero en tres partes: un signo (+, -, o blanco),
 * la parte entera, y la parte decimal.
 */

void separate(double  num,        /* entrada - valor a ser separado */
              char   *signo,      /* salida - signo de num */
              int    *parte_ent,  /* salida - parete entera de num */
              double *parte_dec)  /* salida - parte decimal de num */
{
  double magnitud; /* vaiable local - magnitud de num      */

 /* Determina signo de num */
  if (num < 0)
    *signo = '-';
  else if (num == 0)
    *signo = ' ';
  else
    *signo = '+';
  /* Halla magnitud de num(su valor absoluto) y
     separa la parte entera y decimal */
  magnitud = fabs(num);
  *parte_ent = floor(magnitud);
  *parte_dec = magnitud - *parte_ent;
}


int main(void)
{
  double valor;    /* entrada - numero a analizar */
  char   signo;       /* salida - signo del valor */
  int    part_ent;      /* salida - parte entera del valor */
  double part_dec;       /* salida - parte decimal del valor */

  /* Lee dato  */
  printf("Dar un valor a descomponer> ");
  scanf("%lf", &valor);
  /* Separa el dato en tres partes */
  separate(valor, &signo, &part_ent, &part_dec);
  /* Muestra resultado */
  printf("Partes de %.4f\n  signo: %c\n", valor, signo);
  printf("  parte entera: %d\n", part_ent);
  printf("  parte decimal:  %.4f\n", part_dec);
  return (0);
}
```

Ejemplo 8:

Función recursiva que realiza la multiplicación de dos números enteros positivos mediante la suma acumulado del multiplicando por el número de veces indicado por el multiplicador.

```
  /*
   *  Realiza multiplicacion entera usando el operador +.
   *  Asume n > 0
   */

  int multiplica(int m, int n)
  {
    int res;

    if (n == 1)
      res = m;                        /* caso simple */
    else
      res = m + multiply(m, n  - 1);  /* paso recursivo */
    return (res);
  }
```

Ejemplo 9:

Función que halla las raíces de una ecuación mediante el método de bisección.

Análisis
Requerimientos de datos:

Dato del problema
double x_izq /* punto izquierdo del intervalo */
double x_der /* punto derecho del intervalo */
double epsilon /* tolerancia de error */

Salida del problema
double raiz /* raiz aproximada de f */

Variable del programa
double x_mitad /* punto medio del intervalo */

Diseño
Algoritmo inicial:
1. Si es un caso simple, resolverlo
   1.1. Caso 1: Si el intervalo es menor que epsilon retorna el punto medio
   1.2. Caso 2: Si la función en el punto medio es cero, retorna el punto medio
   sino
   1.3. bisecta el intervalo y ejecuta recursivamente sobre el intervalo mitad que contiene la raíz

Refinamiento del algoritmo paso 1.3:
1.3.1. if f(x_izq) * f(x_der) < 0
1.3.2. Halla raiz mediante bisección(x_izq, x_mitad)
else
1.3.3. Halla raiz mediante bisección(x_mitad , x_der)

```
  /*
   * Metodo de biseccion (version recursiva) para aproximar la raiz de una funcion
f
   * en el intervalo [x_izq, x_der].  Asume que el signo de f(x_izq) y f(x_der)
son
   * diferentes. La aproximacion queda dentro de un epsilon de la raiz.
   */

  double biseccion(double x_izq,   /* entrada - puntos inicial y final */
                   double x_der)      /* para buscar una raiz */
                   double epsilon)    /* entrada - tolerancia de error */
  {
    double raiz,        /* raiz aproximada*/
           x_mitad;     /* punto medio del intervalo */

    /* Calcula punto medio del intervalo */
    x_mitad = (x_izq + x_der) / 2.0;

    if (x_der - x_izq < epsilon)            /* caso 1*/
      raiz = x_mitad;
    else if (f(x_mitad) == 0.0)             /* caso 2 */
      raiz = x_mitad;
    else if (f(x_izq) * f(x_mitad) < 0.0)   /* raiz en [x_izq, x_mitad] */
      raiz = biseccion(x_izq, x_mitad, epsilon);
    else                                    /* raiz en [x_mitad, x_der] */
      raiz = biseccion(x_mitad, x_der, epsilon);
    return (raiz);
  }
```

Ejemplo 10:
Función que devuelve el mayor valor de un array.

```
  /*
```

```
 *  Devuelve el mayor valor de los valores de un array
 *  de dimension n > 0
 */

int halla_max(const int lista[], /* entrada - lista de n enteros */
              int       n)       /* entrada - numero de elementos */
{
  int i,
      max_curso;      /* mayor valor en curso */

  /* Mayor elemento inicial. */
  max_curso = lista[0];
  /* Compara cada elemento restante con el mayor en curso;
     guarda el mayor */
  for  (i = 1;  i < n;  ++i)
    if (lista[i] > max_curso)
      max_curso = lista[i];
  return (max_curso);
}
```

Ejemplo 11:
Programa que imprime una tabla de valores de un array y la dierencia de cada valor con la media de lso mismos. Utiliza una función que calcula la media del array.

```
/*
 * Calcula el promedio de una lista de datos e imprime la
 * diferencia entre cada valor y el promedio.
 */

#include <stdio.h>
/*
 * Calcula el promedio de los n elementos de una lista de numeros (n > 0)
 */
double promedio(const double lista[], /* entrada - lista de nuemros */
                int  n)        /* entrada - numero de elementos a procesar */
{
  int    i;
  double suma = 0;

  for  (i = 0;  i < n;  ++i)
    suma += lista[i];
  return (suma / n);
}

#define MAX_ELEM 8

int main(void)
{
  int    i;
  double x[MAX_ELEM], promed;

  /* Lee datos */
  printf("Enter %d numbers> ", MAX_ELEM);
  for  (i = 0;  i < MAX_ELEM;  ++i)
    scanf("%lf", &x[i]);
  /* Calcula e imprime el promedio */
  promed = promedio(x, MAX_ELEM);
  printf("el promedio es %.2f\n", promed);
  /* Imprime la diferencia entre cada elemento y el promedio */
  printf("\nTabla de diferencias entre los datos y el promedio\n");
  printf("Indice       Elemento      Diferencia\n");
  for  (i = 0;  i < MAX_ELEM;  ++i)
    printf("%3d    %9.2f     %9.2f\n", i, x[i], x[i] - promed);
  return (0);
}
```

Ejemplo 12:
Función que realiza una búsqueda lineal sobre un array para hallar un elemento objetivo especificado como argumento. Devuelve el índice donde se encuentra el elemento o -1 si no se encuentra.

```
  #define NO_ENCONT -1   /*  Valor devuelto por la funcion busca
                                          si el valor objetivo no se halla */
  /*
   *  busca un valor objetivo entre los n elementos de un array arr (n >= 0)
   *  Devuelve el indice de objetivo o NO_ENCONT
   */

  int busca(const int arr[],    /*  array a examinar */
            int       objetivo,   /*  valor buscado */
            int       n)        /*  numero de elementos del array */
  {
    int i,
        hallado = 0,     /*  indicador de si objetivo ha sido hallado  */
        indice;         /*  indice del valor objetivo  */

   /* Compara cada elelento con objetivo */
    i = 0;
    while (!hallado && i < n) {
      if (arr[i] == objetivo)
        hallado = 1;
      else
        ++i;
    }
    /* Devuelve indice del elemento coincidente con objetivo o NO_ENCONT */
    if (hallado)
      indice = i;
    else
      indice = NO_ENCONT;
    return (indice);
  }
```

Ejemplo 12:

Función que calcula la suma de dos arrays que se pasan como argumentos y devuelve el resultado en otro argumento.

```
  /*
   *  Suma los elementos respectivos de los arrays ar1 and ar2,
   *  guardando el resultado en  arsum.
   */

  void suma_arrays(const int ar1[],   /* entrada - */
                   const int ar2[],   /* arrays a sumar*/
                   int       arsum[], /* salida - suma de los elementos ar1 y ar2
*/
                   int       n)       /* entrada - numero de los elementos  */
  {
    int i;

    /* suma elementos correspondientes de los arrays ar1 y ar2  */
    for  (i = 0;  i < n;  ++i)
      arsum[i] = ar1[i] + ar2[i];
  }
```

Ejemplo 13:

Función que realiza un ordenamiento de los elementos de un array (enteros) utilizando el algoritmo de burbuja mejorado con un centinela para detectar si el array ya está ordenado.

Análisis

Requerimientos de datos:

Dato del problema

lista /* array de datos a ordenar */
int n /* numero de elementos del array */

Salida del problema

lista /* array de datos ordenados */

Diseño

Algoritmo inicial:

1. Repetir
   1.1. Examinar cada par de elementos adyacentes del array e intercambiarlos si no están en orden mientras el array no este ordenado

Refinamiento del algoritmo paso 1.1:

1.1. Examinar cada par de elementos adyacentes del array e intercambiarlos si no están en orden
   1.1.1. Inicializar ordenado a 1 (ordenado)
   1.1.2. Repetir para cada par de valores adyacentes
   1.1.3. if el par de valores no estan en orden
   1.1.4. intercambiar los valores

```
/*
 *  Ordena un array de n elementos (n >= 0)
 */

void burbuja(int lista[], /* entrada/salida - array a ordenar */
             int n)      /* entrada - numero de elementos a ordenar */
{
  int i,
      iteracion,  /* numero de la iteracion en curso a traves del array */
      temp,       /* valor temporal usado para intercambio de valores */
      ordenado;   /* indicador si el array esta ordenado */

  iteracion = 1;
  do {
  /* Asume que el array esta ordenado mientras no haya intercambios  */
    ordenado = 1;
  /* Compara todos los elementos del array en una iteracion */
    for  (i = 0;  i < n - iteracion;  ++i) {
      if (lista[i] > lista[i + 1]) {
  /* Intercambio de valores */
        temp = lista[i];
        lista[i] = lista[i + 1];
        lista[i + 1] = temp;
        ordenado = 0;
      }
    }
    ++iteracion;
  } while(!ordenado);
}
```

Ejemplo 14:

Programa que grafica en modo texto los valores de una funcion $t^2 - 4t + 5$ para valores de t en el rango entre 0 a 10. Utiliza una cadena de caracteres en el que coloca un asteriso en el lugar correspondiente al valor de la función.

```
/*                                        2
 *  Dibuja la funcion  f(t) = t  - 4t + 5  para t entre 0 y 10
 */
#include <stdio.h>

#define MAX_VAL 65            /* valor maximo de la funcion*/

/*
 *            2
 *  f(t) = t - 4t + 5
 */
int f(int t)
{
  return (t * t - 4 * t + 5);
```

```
}

int main(void)
{
  char plot[MAX_VAL + 2];               /* linea a imprimir */
  int i, t, valfun;

  /* Imprime cabeceros */
  for  (i = 0;  i <= MAX_VAL;  i += 5)
    printf("%5d", i);
  printf("\n");
  for  (i = 0;  i <= MAX_VAL;  i += 5)
    printf("    |");
  printf("\n");
  /* Inicializa linea a imprimir a blancos */
  for  (i = 0;  i <= MAX_VAL + 1;  ++i)
    plot[i] = ' ';
  /* Calcula e imprime f(t) para cada valor de t desde 0 a 10 */
  for  (t = 0;  t <= 10;  ++t) {
    valfun = f(t);
    plot[valfun] = '*';
    plot[valfun + 1] = '\0';
    printf("t=%2d%s\n", t, plot);
    plot[valfun] = ' ';
    plot[valfun + 1] = ' ';
  }
  return (0);
}
```

Ejemplo 15:

Programa para graficar una función continua en un rango especificado por el usuario. Se asume que pantalla o la zona gráfica tiene 24 filas y 80 columnas.

Análisis

La región del plano x-y que se proyectará en el ordenador está limitada por cuatro líneas:

1. La línea *x=x_inic* a la izquierda
2. La línea *x=x_final* a la derecha
3. La línea *y=y_min*, valor mínimo calculado para *f*(*x*) en el intervalo [*x_inic*, *x_final*]
4. La línea *y=y_max*, valor máximo calculado para *f*(*x*) en el intervalo [*x_inic*, *x_final*]

El paso incremental en *x* para evaluar la función es:

x_incrp = (x_final - *x_inicp) / (NCOLUM - 1)

El valor incremental correspondiente a cada fila en el eje y es:

y_incrp = (ymax - ymin) / (NFILAS - 1)

El número de filas para un valor y se calcula como:

y_dist = (ymax - yv[k]) / *y_incrp;
r = (int)(y_dist + 0.5);

Para almacenar el gráfico de la función se usa una matriz graf[r][k] que se rellena con asteriscos según los valores de la función::

```
/*
 *  Grafica una funcion
 *                            2
 *      f(x) = 4.0 * (x - 0.5)
 *  en una malla de NFILAS filas por NCOLUM columnas
 */

#include <stdio.h>
#include <math.h>

#define NFILAS  20              /* numero de filas en la malla    */
#define NCOLUM  70              /* numero de columnas en la malla */

/*                             2
 * Funcion a graficar 4(x - 0.5)
 */
```

```
  double f(double x)
  {
    return (4.0 * pow(x - 0.5, 2.0));
  }

  /*
   *  Pregunta al usuario los valores inicial y final de x
   *  y calcula el incremento de x
   */

  void lee_datos(double *x_inicp,  /*  salida - valor inicial de x */
                 double *x_incrp)  /*  salida - incremento de x */
  {
    double x_final;

    /*  Lee datos de valores inicial y final de x  */
    printf("La funcion se graficara entre los valores especificados.");
    printf("\nDar valor inicial de X > ");
    scanf("%lf", x_inicp);
    printf("Dar valor final de X > ");
    scanf("%lf", &x_final);
    /*  Calcula y devuelve incremento de x */
    *x_incrp = (x_final - *x_inicp) / (NCOLUM - 1);
  }

/*
   *  Grafica la funcion marcando las celdas apropiadas del array
   *  graf con asteriscos.
   */
  void graf_puntos(char    graf[NFILAS][NCOLUM], /* salida - malla de la grafica
*/
                   double  x_inic,            /* entrada  - valor inicial
de x */
                   double  x_incr,            /* entrada  - incremento x
*/
                   double *y_maxp,            /* salida - mayor valor de y
*/
                   double *y_incrp)           /* salida - incremento y */
  {
    double yv[NCOLUM], y_dist, ymin, ymax, xm;
    int    i, j, r, k;

    /*  Inicializa malla graf con blancos            */
    for  (i = 0;  i < NFILAS;  ++i)
      for  (j = 0;  j < NCOLUM;  ++j)
        graf[i][j] = ' ';
    /* Fase 1:  Determina valores maximo y minimo de la funcion */
    ymax = f(x_inic);
    ymin = ymax;
    yv[0] = ymax;
    for  (k = 1;  k < NCOLUM; ++k) {
      xm = x_inic + k * x_incr;
      yv[k] = f(xm);
      if (yv[k] > ymax)
        ymax = yv[k];
      if (yv[k] < ymin)
        ymin = yv[k];
    }
    /*  Devuelve el valor maximo de y, incremento de y  */
     *y_maxp = ymax;
     *y_incrp = (ymax - ymin) / (NFILAS - 1);
  /*  Fase 2:  Marca graf columna por columna usando valores de la funcion */
    for  (k = 0;  k < NCOLUM;  ++k) {
      y_dist = (ymax - yv[k]) / *y_incrp;
      r = (int)(y_dist + 0.5);
      graf[r][k] = '*';
    }
  }

  #define INTERVALO_ESCALA 10  /* num. de columnas en etiquetas de esc. x */

  /*
   * Imprime graf con etiquetas
   */
```

```
  void impr_graf(char graf[NFILAS][NCOLUM], /* entrada -malla de caract. en graf
*/
                   double     x_inic,      /* entrada - valor inicial de x */
                   double     x_incr,      /* entrada - incremento de x */
                   double     y_max,       /* entrada - mayor valor de y */
                   double     y_incr)      /* entrada - incremento y */
  {
    int r, k, i;
    double x, y,
    x_escala[NCOLUM / INTERVALO_ESCALA]; /* etiq. para esc. x debajo de graf */

    /*  Imprime graf fila por fila con etiquetas a la izquierda de la primera a
        cada cinco filas */
    printf("\n   Y\n");
    printf("%6.1f   ", y_max);   /*  etiqueta e imprime fila inicial */
    for (k = 0;  k < NCOLUM; ++k)
      printf("%c," graf[0][k]);
    printf("\n");
    for  (r = 1;  r < NFILAS;  ++r) {  /* imprime resto de graf    */
      if ((r + 1) % 5 == 0) {      /* etiqueta cada cinco filas */
        y = y_max - r * y_incr;
        printf("%6.1f   ", y);
      }
      else {
        printf("         ");
      }
      for (k = 0;  k < NCOLUM; ++k)
        printf("%c", graf[r][k]);
      printf("\n"),
      }
    /*  Calcula valores de escala x */
    k = INTERVALO_ESCALA;
    for  (i = 0;  i < NCOLUM / INTERVALO_ESCALA;  ++i) {
      x = x_inic + (k - 1) * x_incr;  /* valor de x para la columna k */
      x_escala[i] = x;
      k += INTERVALO_ESCALA;
    }
    /*  Imprime la escala x al final de graf  */
    printf("         |--------|");
    for  (i = 0; i < NCOLUM / INTERVALO_ESCALA - 1;  ++i)
      printf("---------|");
    printf("\n     %6.1f", x_inic);
    for  (i = 0;  i < NCOLUM / INTERVALO_ESCALA;  ++i)
      printf("   %6.1f ", x_escala[i]);
    printf("\n                                          X-->\n");

  }

  int main(void)
  {
    double x_inic, x_incr, y_max, y_incr;
    char graf[NFILAS][NCOLUM];

    /*  Lee datos, grafica funcion e imprime resultados  */
    lee_datos(&x_inic, &x_incr);
    graf_puntos(graf, x_inic, x_incr, &y_max, &y_incr);
    impr_graf(graf, x_inic, x_incr, y_max, y_incr);
    return (0);
  }
```

Ejemplo 16:

Función que calcula el producto de dos matrices que se pasan como argumentos y devuelve el resultado en otro argumento. Cada elemento $C_{i,j}$ de la matriz resultante se calcula aplicando la siguiente fórmula:

$$C(i,j) = \sum_{k=1}^{N} A(i,k) * B(k,j)$$

```
  /*
   *  Multiplica matrices A y B produciendo matriz producto C
   */

  void mat_prod(double c[M][P],  /* salida - matriz producto M x P  */
```

```
                double a[M][N],  /* entrada  - matriz M x N         */
                double b[N][P])  /* entrada  - matriz N x P        */
  {
    int i, j, k;

    for  (i = 0;  i < M;  ++i) {
      for  (j = 0;  j < P;  ++j) {
        c[i][j] = 0;
        for  (k = 0;  k < N;  ++k)
          c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
```

Ejemplo 17:

Programa que define un tipo de dato complejo e implementa un conjunto de operaciones para el mismo, como leer, imprimir, valor absoluto, sumar, restar y multiplicar números complejos.

```
  /*
   *  Operadores para procesar numeros complejos
   */

  #include <stdio.h>
  #include <math.h>

  /*  Tipo de dato para definir un numero complejo */
  typedef struct {
    double real, imag;
  } complejo_t;

  /*
   *  Funcion de entrada para leer un numero complejo
   * 1 => dato valido, 0 => error, negativo o EOF => final fichero
   */
  int lee_complejo(complejo_t *c)  /* salida - direccion de una variable compleja
*/
  {
    int estado;

    estado = leef("%lf%lf", &(*c).real, &(*c).imag);
    if (estado == 2)
      estado = 1;
    else if (estado != EOF)
      estado = 0;
    return (estado);
  }

  /*
   *  funcion de impresion de un complejo como (a + jb) o (a - jb)
   */
  void print_complejo(complejo_t c)    /* entrada - numero complejo a imprimir */
  {
    double a, b;
    char   signo;

    a = c.real;
    b = c.imag;
    printf("(");
    if (fabs(a) < .005 && fabs(b) < .005) {
      printf("%.2f", 0.0);
    }
    else if (fabs(b) < .005) {
      printf("%.2f", a);
    }
    else if (fabs(a) < .005) {
      printf("j%.2f", b);
    }
    else {
      if (b < 0)
        signo = '-';
      else
        signo = '+';
      printf("%.2f %c j%.2f", a, signo, fabs(b));
```

```
    }
    printf(")");
  }

  /*
   *  Devuelve la suma de los valores complejos c1 y c2
   */
  complejo_t suma_complejo(complejo_t c1, complejo_t c2)
  {
    complejo_t csuma;

    csuma.real = c1.real + c2.real;
    csuma.imag = c1.imag + c2.imag;
    return (csuma);
  }

  /*
   *  Devuelve la diferencia c1 - c2
   */
  complejo_t resta_complejo(complejo_t c1, complejo_t c2)
  {
    complejo_t cdiff;

    cdiff.real = c1.real - c2.real;
    cdiff.imag = c1.imag - c2.imag;
    return (cdiff);
  }

/*
   *  Devuelve el producto de c1 y c2
   */

  complejo_t multiplica_complejo(complejo_t c1, complejo_t c2)
  {
    complejo_t cmult;

    cmult.real = c1.real*c2.real - c1.imag*c2.imag;
    cmult.imag = c1.real*c2.imag + c2.real*c1.imag;
    return (cmult);
  }

  /*  ** Conductor **
   *  Devuelve division de los complejos (c1 / c2)
   */
  complejo_t divide_complejo(complejo_t c1, complejo_t c2)
  {
    printf("Funcion que devuelve la division compleja. Por ahora devuelve c1\n");
    return (c1);
  }

  /*
   *  Devuelve el valor absoluto del complejo c
   */
  complejo_t abs_complejo(complejo_t c)
  {
    complejo_t cabs;

    cabs.real = sqrt(c.real * c.real + c.imag * c.imag);
    cabs.imag = 0;
    return (cabs);
  }

  int main(void)
  {
    complejo_t com1, com2;

    /*  Lee dos numeros complejos */
    printf("Da la parte real e imaginaria de un numero complejo\n");
    printf("separados por un espacio> ");
    lee_complejo(&com1);
    printf("Da un segundo numero complejo> ");
    lee_complejo(&com2);
    /*  Suma e imprime la suma */
    printf("\n");
    print_complejo(com1);
```

```
    printf("  +  ");
    print_complejo(com2);
    printf("  =  ");
    print_complejo(suma_complejo(com1, com2));
    /*  Resta e imprime la diferencia */
  printf("\n\n");
    print_complejo(com1);
    printf("  -  ");
    print_complejo(com2);
    printf("  =  ");
    print_complejo(resta_complejo(com1, com2));
    /*  Imprime el valor absoluto de un complejo */
    printf("\n\n|");
    print_complejo(com1);
    printf("|  =  ");
    print_complejo(abs_complejo(com1));
    printf("\n");
    return (0);
  }
```

Ejemplo 18:

Programa que calcula el perímetro de un polígono definido por una serie de puntos almacenados en un fichero. El programa calcula además el procentaje de la longitud de cada arista respecto del total y el porcentahe acumulado de los mismos.

Análisis

Requerimientos de datos:

Datos del problema

Fichero de entrada: datos de coordenadas
Fichero de salida: longitudes de las aristas y porcentajes
Tipo de dato estructurado para los puntos: punto_t
MAX 100 /* tamaño de los arrays */
punto_t punto[MAX] /* puntos del perimetro */

Salida del problema

```
double  longitud[MAX],  /*  longitud del contorno  */
        porc_per[MAX],    /*  porcentajes de perimetro */
        cum_porc_per[MAX], /*  porcentajes de perimetro acumulado*/
        perim;      /*  longitud perimetro  */
```

Fórmula de la distancia entre dos puntos:

$$distancia = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Diseño

Algoritmo inicial:

1. Lee y contar los puntos desde el fichero de datos
2. Calcular la longitud de cada arista y la longitud total
3. Calcular los porcentajes de la longitud de cada arista respecto al total y el porcentaje acumulado
4. Imprimir los resultados en una tabla y en un fichero

```
/*
 *  Programa que lee una serie de puntos de un contorno
 *  y calcula la distancia entre puntos adyacentes. Se imprime las
 *  coordenadas de los puntos extremos, las longitudes de cada linea,
 *  y sus porcentajes respecto del total del perimetro asi como los
 *  porcentajes acumulados en un fichero de salida. Al mismo tiempo se
 *  imprime en pantalla como tabla.
 */
#include <stdio.h>
#include <math.h>
```

```
typedef struct {
         double x, y;
   } punto_t;

#define LONG_CAD 80
/*
 *  Pregunta al usuario por el nombre del fichero texto que contiene
 *  los datos de puntos y devuelve el numero de puntos leidos en el
 *  parametros np.
 */
void
lee_puntos(int      max,   /* entrada - maximo puntos  */
           punto_t  punto[], /* salida - array de datos        */
           int     *np)    /* salida - numero de puntos datos  */
{
  int     i, estado;
  punto_t pto;
  char    nomb_fich[LONG_CAD];
  FILE   *filep;

/*  Lee nombre del fichero de datos y lo abre */
  printf("Dar nombre del fichero que contiene los puntos> ");
  scanf("%s", nomb_fich);
  filep = fopen(nomb_fich, "r");
  if (filep == NULL) { /* si no se abre el fichero emite mensaje
                          de error y devuelve cero como numero de ptos   */
       printf("No se puede abrir fichero %s\n", nomb_fich);
       *np = 0;
  } else { /* el ciclo termina con EOF, error, o si se llena el array  */
    i = 0;
    for  (estado = fscanf(filep, "%lf%lf", &pto.x, &pto.y);
          estado == 2  &&  i < max;
          estado = fscanf(filep, "%lf%lf", &pto.x, &pto.y)) {
            punto[i++] = pto;
          }
     /* Mensaje de error en caso de salida prematura */
     if (estado > EOF  &&  estado < 2) {
         printf("*** Error en formato de datos ***\n");
         printf("*** Usando primeros %d puntos ***\n", i);
     } else if (estado == 2) {
         printf("*** Error: mas de %d puntos en fichero %s ",
                max, nomb_fich);
         printf("***\n*** Solo se usan %d puntos ***\n", i);
     }
     /* Devuelve el tamaño del array      */
     *np = i;
  }
}

/*
 *  Calcula la distancia entre los puntos inicial y final.
 */
double
distancia(punto_t inicial, punto_t final) /* entrada - puntos extremos de una linea
*/
{
 double xdiff, ydiff;

 xdiff = final.x - inicial.x;
 ydiff = final.y - inicial.y;
 return (sqrt(xdiff * xdiff + ydiff * ydiff));
}

/*
 *  Calcula la longitud de cada linea y el perimetro total.
 */
void
calc_longits(const punto_t punto[],  /*  entrada - lista de puntos   */
             int           n,        /*  entrada - numero de puntos  */
             double        longit[], /*  salida - longitud de cada linea */
             double       *perim)    /*  salida - perimetro total */
{
 int i;

 /*  Calculo de la longitud de cada linea y lo suma al total */
```

```
 *perim = 0;
 for  (i = 0;  i < n - 1;  ++i) {
   longit[i] = distancia(punto[i], punto[i + 1]);
   *perim += longit[i];
 }
 longit[n - 1] = distancia(punto[n - 1], punto[0]);
 *perim += longit[n - 1];
}

/*
 *  Calcula la longitud porcentual de cada linea respecto del permitro total
 *  y el porcentaje acumulado.
 */
void
calcula_porc_per(const double longit[],  /*  entrada - longitud de cada linea */
            int          n,          /*  entrada - numero de puntos */
            double       perim,      /*  entrada - longitud del perimetro */
            double       pct[],      /*  salida - pocentaje de cada longit */
            double       cum_pct[]) /*  salida - porcentaje acumulado */
{
 int    i;
 double tot_pct = 0;

 for  (i = 0;  i < n;  ++i) {
   pct[i] = longit[i] / perim * 100.0;
   tot_pct += pct[i];
   cum_pct[i] = tot_pct;
 }
}

/*
 *  Imprime los resultados en la pantalla y en un fichero texto.
 */
void
escribe_result(const punto_t punto[],  /*  entrada - puntos del contorno */
               const double  longit[], /*  entrada - longitudes de las lineas */
               const double  pct[],     /*  entrada - porcentaje de la long. de
cada linea */
               const double  cum_pct[], /*  entrada - % acumulado del total */
               int           n,         /*  entrada - numero de puntos del contorno
*/
               double        perim)     /*  entrada - long. total del contorno */
{
 char  nomb_fich[LONG_CAD];  /* nombre del fichero de salida */
 FILE *outp;                /*  puntero al fichero de salida  */
 int   i;
/*  Lee y abre fichero de salida */
 printf("Dar nombre del fichero desalida para colocar resultados\n> ");
 scanf("%s", nomb_fich);
 outp = fopen(nomb_fich, "w");
/*  Muestra y guarda cabercera de la tabla                    */
 printf("\n           Calculo de la longitud de un contorno\n\n");
 fprintf(outp,
         "           Calculo de la longitud de un contorno\n\n");
 printf("Numero de puntos %d.\n", n);
 fprintf(outp, "Numero de puntos %d.\n", n);
 printf("Longitud del contorno = %.3f m.\n\n", perim);
 fprintf(outp, "Longitud del contorno = %.3f m.\n\n", perim);
 printf("Linea        Punto      Longitud      %% acumulado\n");
 fprintf(outp, "Linea        Punto      Longitud      %% acumulado\n");
 printf("Numero     x          y       (en m)     Contorno      %%\n\n");
 fprintf(outp, "Numero     x          y       (en m)     Contorno      %%\n\n");
/*  Muestra y guarda la tabla de resultados */
 for  (i = 0;  i < n;  ++i) {
   printf("%3d%12.3f%12.3f%12.3f%12.3f%12.3f\n", i + 1, punto[i].x,
   punto[i].y, longit[i], pct[i], cum_pct[i]);
   fprintf(outp, "%3d%12.3f%12.3f%12.3f%12.3f%12.3f\n", i + 1, punto[i].x,
   punto[i].y, longit[i], pct[i], cum_pct[i]);
 }
/*   Cierra fichero fuente   */
 fclose(outp);
}

#define MAX 100
```

```
int main(void)
{
        int     n;            /*  numero de puntos  */
        punto_t punto[MAX];     /*  punto de las aristas  */
        double  longitud[MAX],  /* longitud del contorno */
                porc_per[MAX],     /*  porcentajes de perimetro */
                cum_porc_per[MAX], /*  porcentajes de perimetro acumulado*/
                perim;        /*  longitud perimetro   */

        /*  Obtiene puntos  */
        lee_puntos(MAX, punto, &n);
        /*  Calcula longitud de cada arista y longitud total */
        calc_longituds(punto, n, longitud, &perim);
        /*  Calcula procentaje de cada arista respecto al perim. total */
        compute_porc_per(longitud, n, perim, porc_per, cum_porc_per);
        /*  Imprime fichero de salida y tabla         */
        escribe_result(punto, longitud, porc_per, cum_porc_per, n, perim);
        return(0);
  }
```

Ejemplo 19:

Programa que aproxima las raices de las funciones *f* y *g* mediante el método de Newton.

El método de Newton requiere una de un valor inicial de la raíz x_0 generando raíces aproximadas x_1, $x_2,\ldots, x_j, x_{j+1}$ usando la fórmula iterativa:

$$x_{j+1} = x_j - \frac{f(x_j)}{f'(x_j)}$$

donde es la derivada de la función *f* evaluada en el punto x= x_j. El criterio de convergencia del método es el siguiente:

$$\left|x_j - x_{j-1}\right| < \varepsilon$$

Si el método no es capaz de aproximar la raíz según el criterio anterior se aplica un número máximo de iteraciones.

Nótese que en la implementación de la función se pasan funciones como argumentos de la función.

```
  /*
       *  Halla las raices de las ecuaciones
       *       f(x) = 0  y  g(x) = 0
       *  dado una condicion inicial por el usuario y aplicando el metodo
       *  de Newton a partir de una libreria numerica personal.
       */

      #include <stdio.h>
      #include <math.h>
      #include "newton.h"  /* Fichero cabecera del metodo de newton */
      /*  Funciones para las que se hallaran la raiz y sus derivadas */
      /*     3      2
       *  5x  - 2x  + 3
       */
      double f(double x)
      {
            return (5 * pow(x, 3.0) - 2 * pow(x, 2.0) + 3);
      }
      /*       2
       *  15x  - 4x
       */
      double f_deriv(double x)
      {
            return (15 * pow(x, 2.0) - 4 * x);
      }
      /*     3      2
       *  8x  - 6x  - 12x
       */
      double g(double x)
      {
            return (8 * pow(x, 3.0) - 6 * pow(x, 2.0) - 12 * x);
      }
      /*       2
       *  24x  - 12x - 12
```

```
      */
     double g_deriv(double x)
     {
           return (24 * pow(x, 2.0) - 12 * x - 12);
     }

     int main(void)
     {
           double raiz_ini,  /* raiz inicial dada por el usuario */
                  epsilon,   /* tolerancia de error */
                  raiz;
           int    error;

           /*  Lee raiz inicial y tolerancia de error para f  */
           printf("\nDar raiz inicial para la funcion f> ");
           scanf("%lf", &raiz_ini);
           printf("\nDar tolerancia de error> ");
           scanf("%lf", &epsilon);
           /*  Usa metodo de Newton para buscar una raiz de f */
           printf("\nFunction f\n");
           raiz = newton(f, f_deriv, raiz_ini, epsilon, &error);
           if (error)
                 printf("No hay raiz de f para la raiz inicial %.7f\n", raiz_ini);
           else
                 printf("   f(%.7f) = %e\n", raiz, f(raiz));
           /*  Lee raiz inicial y tolerancia de error para g  */
           printf("\nDar raiz inicial para la funcion g> ");
           scanf("%lf", &raiz_ini);
           printf("\nDar tolerancia de error> ");
           scanf("%lf", &epsilon);
           /*  Usa metodo de Newton para buscar una raiz de g */
           printf("\nFuncion g\n");
           raiz = newton(g, g_deriv, raiz_ini, epsilon, &error);
           if (error)
               printf("No hay raiz de f para la raiz inicial %.7f\n", raiz_ini);
           else
                 printf("   g(%.7f) = %e\n", raiz, g(raiz));
           return (0);
     }
  }
```

newton.c

```
  #include <stdio.h>
  #include <math.h>

     #define TRAZA
     #define FALSE      0
     #define TRUE       1
     #define MAX_ITER 100  /* maximo de iteraciones permitidas */
     /*
      *  Implementacion del metodo de Newton para hallar la raiz de una funcion f.
      *  Halla la raiz y devuelve en el parametro salida a FALSE,
      *  devuelve TRUE en salida si fderiv devuelve un valor cero o si el
      *  metodo no converge en MAX_ITER iteraciones
      */
     double
     newton(double  f(double farg),       /* entrada  - funcion             */
            double  fderiv(double fdarg), /* entrada  - derivada de la funcion
*/
            double  inicial,        /* entrada  - raiz inicial  */
            double  epsilon,      /* entrada  - tolerancia a error  */
            int    *errp)         /* salida - indicador de error */
     {
           double previo,        /* aproximacion previa   */
                  aprox,         /* nueva aproximacion    */
                  deriv_val;     /* valor de la derivada  */
           int    num_iter = 0;  /* numero de iteraciones en curso   */
           *errp = FALSE;

           aprox = inicial;
           do {
               ++num_iter;
               previo = aprox;
               deriv_val = fderiv(previo);
```

```
            if (num_iter > MAX_ITER)
                  *errp = TRUE;
            else if (deriv_val == 0)
                  *errp = TRUE;
            else
                  aprox = previo - f(previo) / deriv_val;
            /* Imprime resultados si TRAZA se activa        */
            #if defined(TRAZA)
                    if (*errp)
                          printf("la aproximacion no converge\n");
                    else
                          printf("%3d. Valor de la Funcion en %.7f = %e\n",
                                 num_iter, aprox, f(aprox));
            #endif
          } while (!(*errp)&& fabs(aprox - previo) >= epsilon);
          return (aprox);
    }
```

header.h

```
  double
     newton(double  f(double farg),       /* entrada  - funcion             */
            double  fderiv(double fdarg), /* entrada  - derivada de la funcion
*/
            double  inicial,        /* entrada  - raiz inicial  */
            double  epsilon,      /* entrada  - tolerancia a error  */
            int    *errp);         /* salida - indicador de error */
```

Ejemplo 20:

Función que calcula los coeficientes de regresión A y B para una colección de N puntos (X_i, Y_i) dando como datos el valor promedio de X e Y. El cálculo se realiza aplicando las siguientes fórmulas:

$$B = \frac{\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{N}(X_i - \overline{X})^2}$$

$$A = \overline{Y} - B * \overline{X}$$

```
  /*
     *  Calcula los coeficientes de regresion lineal a y b de n puntos de
     *  datos (x[i], y[i]), dados x_prom y y_prom.
     */
    void
    regresion_lineal(const double x[],   /* entrada - arrays de coordenadas (x,y)
                     const double y[],    /*         of n puntos de datos  */
                     int          n,      /* entrada - numero de puntos datos  */
                     double       x_prom, /* entrada - promedio de los valores x*/
                     double       y_prom, /* entrada - prom. de los valores y */
                     double *ap,       /* salida - coef. de la regresion lineal */
                     double *bp)
    {
          double sum_xy, sum_xds, x_diff;
          int    i;

          sum_xy = 0;
          sum_xds = 0;
          for  (i = 0;  i < n;  ++i) {
              x_diff = x[i] - x_prom;
              sum_xy += x_diff * (y[i] - y_prom);
              sum_xds += x_diff * x_diff;
          }
          *bp = sum_xy / sum_xds;
          *ap = y_prom - *bp * x_prom;
    }
```

Ejemplo 21:

Función que calcula el coeficiente de correlación entre los valores de dos vectores X e Y que representan los valores de N puntos (X_i, Y_i). El cálculo se realiza aplicando las siguientes fórmulas:

$$r = \sum_{i=1}^{N} \frac{(Z_{X_i} * Z_{Y_i})}{N-1}$$

donde:

$$Z_{X_i} = \frac{(X_i - \overline{X})}{S_X} \qquad Z_{Y_i} = \frac{(Y_i - \overline{Y})}{S_Y}$$

la desviación estándar *S* se calcula previamente y se pasa como argumentos. Su cálculo se realiza según la siguiente fórmula:

$$S_X = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \overline{X})^2}{N-1}}$$

```
/*
    *  Calcula el coeficiente de corelacion r de n
    *  puntos (x[i], y[i])
    */
   double
   r(const double x[],     /* entrada - coordenadas (x, y) de n */
     const double y[],     /*           puntos datos            */
     int          n,       /* entrada - numero de puntos datos  */
     double       x_prom,  /* entrada - promedio de x           */
     double       y_prom,  /*           y valores y             */
     double       sx,      /* entrada - desviacion standard     */
     double       sy)      /*           de los valores x e y    */
   {
         double ps, zxi, zyi;
         int i;

         ps = 0;
         for  (i = 0;  i < n;  ++i) {
             zxi = (x[i] - x_prom) / sx;
             zyi = (y[i] - y_prom) / sy;
             ps += zxi * zyi;
         }
         return (ps / (n - 1));
   }
```

Ejemplo 22:

Función que calcula la integral de una función que se pasa como argumento. El cálculo se realiza aplicando la siguiente fórmula:

$$\int_a^b f(z)\,dz = \frac{h}{3}\left(f(a) + f(b) + 4 \sum_{\substack{i=2 \\ Paso2}}^{n} f(z_{i-1}) + 2 \sum_{\substack{i=2 \\ Paso2}}^{n-2} f(z_i) \right)$$

```
/*
    *  Calcula aproximadamente la integral definida de la funcion f
    *  de a a b usando la regla de Simpson con n intervalos
    */
   double
   simpson(double f(double farg),  /*  entrada - funcion a integrar      */
           double a,               /*  entrada - puntos extremos del     */
           double b,               /*            intervalo [a, b]        */
           int    n)               /*  entrada - numero de subintervalos */
   {
         double h, sum_impar, sum_par;
         int i;

         /*  Calcula tamaño del intervalo     */
         h = (b - a) / n;
```

```
            /*  Calcula sum_impar  */
            sum_impar = 0;
            for  (i = 2;  i <= n;  i += 2) {
                sum_impar  += f(a + (i - 1) * h);
            }
            /*  Calcula sum_par           */
            sum_par = 0;
            for  (i = 2;  i <= n - 2;  i += 2) {
                sum_par += f(a + i * h);
            }
            /*  Retorna aproximacion       */
            return (h / 3.0 * (f(a) + f(b) + 4.0 * sum_impar + 2.0 * sum_par));
      }
```

Ejemplo 23:

El radio medio geométrico (RMG) de una serie de conductores dispuestos paralelamente está definido por la siguiente fórmula:

$$RMG = \sqrt[N^2]{\prod_{k=1}^{N}\prod_{m=1}^{N} D_{km}}$$

Donde: D_{km} = Distancia del conductor *k* al conductor m, siendo $D_{kk} = r_k = e^{-1/4} \cdot r$ donde *r* es el radio del conductor *k*. La distancia entre dos puntos ($P_1=(x_1,y_1)$, $P_2=(x_2,y_2)$) es:

$$D_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Se pide escribir:

1. Una función que calcule la distancia entre dos puntos.
2. Un programa para calcular el radio medio geométrico (RMG) a partir de la lectura de los siguientes datos:
   - leer el número de conductores N.
   - para cada conductor, leer las coordenadas de sus centros (x_k, y_k) y el radio r_k

   Se debe usar la función definida en 1. para el cálculo de las distancias entre conductores. El programa pedirá otra serie de datos mediante la pregunta: Desea continuar (s/n)?, ante los cual el usuario responderá con el carácter s o n.

```
/* Programa que calcula el Radio Medio Geometrico
   de una serie de conductores
*/
  #include <stdio.h>
  #include <math.h>

  #define MAX_COND 100
  struct cond {
   float x;
   float y;
   float r;
  };

  double distancia(struct cond x, struct cond y);

  main()
  {
   struct cond datos[MAX_COND];
   int i,j,nc;
   char c;
   double prod,RMG,ncd;

   do
   {
    printf("Ingresa numero de conductores: ");
    scanf("%d",&nc);
    for (i=0;i<nc;i++)
    {
     printf("Ingresa coordenadas x,y y radio: ");
     scanf("%f %f %f",&datos[i].x,&datos[i].y,&datos[i].r);
```

```
    }
    prod=1.0;
    for (i=0;i<nc;i++)
     for (j=0;j<nc;j++)
     {
       if (i!=j)
         prod*= distancia(datos[i],datos[j]);
       else
         prod*= exp(-0.25)*datos[j].r;
     }
    RMG=exp(1.0/(nc*nc)*log(prod));
    printf("El RMG es: %f\n",RMG);
    printf("desea continuar? (s/n)");
    scanf(" %c",&c);
   } while (c=='s'|| c=='S')
  return(0);
 }

 double distancia(struct cond p1, struct cond p2)
 {
  double d;

  d=sqrt(pow(p1.x-p2.x,2)+pow(p1.y-p2.y,2));
  return(d);
 }
```

Ejemplo 24:

Se tiene un árbol binario de empleados, ordenado según el campo edad, con la siguiente estructura de nodos:

```
struct empleado {
        char *nombre;
        int edad;
        }
struct arbol_empleados {
        struct empleado *info;
        struct arbol_empleados *izq;
        struct arbol_empleados *der;
        }
```

Escribir una función de recorrido del árbol (indicando el tipo) para imprimir el contenido de sus nodos (una línea por cada nodo) en orden ascendente a la edad del empleado. Se da como dato de partida una variable raíz que apunta a la raíz del árbol binario.

```
struct empleado {
   char *nombre;
   int  edad;
};

struct arbol_empleados {
   struct empleado *info;
   struct arbol_empleados *izq;
   struct arbol_empleados *der;
};

/* imprime arbol en in-orden */
void imprimir_in_orden(struct arbol_empleados *p)
{
  if (p!=NULL) {
     imprimir_in_orden(p->izq);
     printf("%s  %d\n",p->info->nombre,p->info->edad);
     imprimir_in_orden(p->der);
}
```

Ejemplo 25:

Una forma de interpolar una serie de N puntos (x_i, y_i) es hacer pasar por ellos un polinomio (de grado N-1). La solución clásica es usar la fórmula de Lagrange que expresada matemáticamente es:

$$p(x)= \sum_{1\le j\le N} \left(\prod_{\substack{1\le i\le N \\ i\neq j}} \frac{x - x_i}{x_j - x_i} \right) y_j$$

Se pide:

a) Escribir una función que realice la interpolación.

b) Escribir una función principal que:
   - lea el número de puntos N.
   - lea los puntos a interpolar (x_1, y_1), (x_2, y_2),....,(x_n, y_n)
   - lea el valor x a interpolar

   A continuación hará uso de la función desarrollada en a) para realizar la interpolación. El programa seguirá pidiendo valores a interpolar mediante la pregunta Desea continuar (s/n)?, ante los cual el usuario responderá con el carácter s o n. En caso afirmativo se pedirá el valor x a interpolar.

```
/* Programa que usa la interpolacion de lagrange
   en una serie de n puntos dados (xi,yi) para
   calcular el valor interpolado de un valor dado
   x. el programa continua preguntando por un valor
   x hasta que el usuario desee abandonar el programa
*/
#include <stdio.h>

#define MAX_PTOS   20

int main(void)
{
 int i,n_ptos;
 float xv, x[MAX_PTOS], y[MAX_PTOS];
 char c;
 float lagrange(int nptos, float x[], float y[], float xv);

 printf("Numero de puntos a interpolar: (Maximo %d) ",MAX_PTOS);
 scanf("%d",&n_ptos);
 for(i=0; i<n_ptos; i++)
 {
  printf("Dar punto %d (x y) ",i);
  scanf("%f%f",&x[i],&y[i]);
 }
 do
 {
  printf("Dar valor a interpolar: ");
  scanf("%f",&xv);
  printf("El valor interpolado es %f\n ",lagrange(n_ptos,x,y,xv));
  printf("Desea continuar (s/n): ");
  scanf(" %c",&c);
 } while (c=='s' || c=='S');
 return 0;
}

/* Funcion para la interpolacion de una serie de
   n puntos (xi,yi) usando el metodo de lagrange
*/
float lagrange(int nptos, float x[], float y[], float xv)
{
 int i,j;
 float sumat=0.0, product;

 for(i=0; i<nptos; i++)
 {
  product=1.0;
  for(j=0; j<nptos; j++)
    if (i!=j)
      product*=(xv - x[j])/(x[i] - x[j]);
  sumat += product*y[i];
 }
 return(sumat);
}
```

Ejemplo 26:

Una serie de Fourier expresa una función f(x) definida en el intervalo $-\pi \le x \le \pi$ en términos de una serie trigonométrica de la forma:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos nx + b_n sin\, nx \right)$$

Si se dan como datos dos vectores a y b de n+1 y n componentes, respectivamente. Se pide:
a) Escribir una función que devuelva el valor de la serie para un valor x que se da como parámetro.
b) Escribir una función principal que:
- lea desde un fichero (datos.dat) lo siguiente:
- el número de componentes n de los vectores. Se considera un valor máximo de 50 componentes.
- los componentes de los vectores a y b, en ese orden.
- lea (del teclado) el valor x para el cálculo de la serie.

A continuación hará uso de la función desarrollada en a) para calcular la serie. El programa seguirá pidiendo valores a calcular mediante la pregunta *Desea continuar (s/n)?*, ante los cual el usuario responderá con el carácter *s* o *n.* En caso afirmativo se pedirá otro valor de x con el que se revaluará el valor de la serie.

```
/* Programa para el calculo de una serie de Fourier
   para los coeficientes leidos desde un fichero y
   para un valor x.
*/
#include <stdio.h>
#include <math.h>

#define MAX 50

double fourier(float a[], float b[], int n, int x);

main()
{
 int i,n;
 float a[MAX], b[MAX],x;
 char c;
 FILE *fp;

 fp=fopen("fourier.dat","r");
 fscanf(fp,"%d",&n);
 if (n>MAX)
 {
   printf("Error. Demasiados puntos\n");
   return -1;
 }
 for (i=0;i<=n;i++)
   fscanf(fp,"%f",&a[i]);
 for (i=1;i<=n;i++)
   fscanf(fp,"%f",&b[i]);
 do
 {
  printf("\nDar valor: ");
  scanf("%f",&x);
  printf("Valor de la serie para %f = %f\n",x,fourier(a,b,n,x));
  printf("Desea continuar (s/n): ");
  scanf(" %c",&c);
 } while (c=='s' || c=='S');
 return 0;
}

/* Funcion para calcular el valor de una serie de
   Fourier dados los coeficientes y el valor x
*/
double fourier(float a[], float b[], int n, int x)
{
 int i;
 double suma=a[0]/2.0;

 for (i=1;i<=n;i++)
   suma+=(a[i]*cos(i*x) - b[i]*sin(i*x));
 return (suma);
```

```
}
```

Ejemplo 27:

Una imagen (tipo mapa de bits) se considera como una matriz de NxM pixels.Un pixel tiene asignado un nivel de gris, que es un número entero sin signo en el rango de 0 a 255 (escala de grises). Una operación para eliminar el ruido en una imagen es la de suavizado, que consiste en asignar el valor de intensidad de cada pixel como el valor medio de la intensidad de los pixels en un entorno del pixel (p) incluído él mismo. Para entornos cuadrados la media resulta de aplicar:

$$f'(x,y) = \frac{1}{n^2} \sum_{i=-p}^{p} \sum_{j=-p}^{p} f(x+i, y+j)$$

donde: n = 2*p + 1 (entorno de nxn pixels)
p = 1, 2, 3, …

En los bordes hay que asegurar que los índices de los pixels de la imagen estén en el rango correcto y dividir sólo por el número de pixels que cumplen tal condición. Se pide:
Escribir una función que reciba como datos una matriz NxN de pixels y un valor entero p, entre 1 y 10, correspondiente al entorno de pixels. la función devolverá otra matriz cuyos elementos sean el resultado de la operación de suavizado.

```
#include <stdio.h>
#define MAX_PIXELS 100
#define MAX_INTPIX 255

int imagen[MAX_PIXELS][MAX_PIXELS];
int imagenf[MAX_PIXELS][MAX_PIXELS];
int n;

main()
{
  void lee_imagen_fichero(int *n, int g[][MAX_PIXELS]);
  void imprime_imageni(int n, int g[][MAX_PIXELS]);
  void imprime_imagenf(int n, float g[][MAX_PIXELS]);
  void suaviza(int p,int n, int g[][MAX_PIXELS], float h[][MAX_PIXELS]);

  lee_imagen_fichero(&n,imagen);
  imprime_imageni(n,imagen);
  suaviza(1,n,imagen,imagenf);
  imprime_imagen(n,imagenf);
  return(0);
}

void lee_imagen_fichero(int *n, int imagen[][MAX_PIXELS])
{
  FILE *fp;
  int i,j;
  int a;

  fp=fopen("imagen.dat","r");
  fscanf(fp,"%d",n);
  for(i=0;i<*n;i++)
   for(j=0;j<*n;j++)
   {
    fscanf(fp,"%d",&a);
    imagen[i][j]=a;
   }
}

void imprime_imageni(int n, int imagen[][MAX_PIXELS])
{
  int i,j;

  printf("Numero de pixels: ");
    printf("%d \n",n);
  printf("Imagen:\n");
  for(i=0;i<n;i++)
  {
```

```
    for(j=0;j<n;j++)
      printf("%6d ",imagen[i][j]);
    printf("\n");
   }
  }

  void imprime_imagenf(int n, float imagen[][MAX_PIXELS])
  {
    int i,j;

    printf("Numero de pixels: ");
      printf("%d \n",n);
    printf("Imagen:\n");
    for(i=0;i<n;i++)
    {
     for(j=0;j<n;j++)
       printf("%6.2f ",imagen[i][j]);
     printf("\n");
  }
  }

  void suaviza(int p,int n, int imagen[][MAX_PIXELS],float imagenp[][MAX_PIXELS])
  {
    int i,j,pix,x,y;
    float sum;

   for(x=0;x<n;x++)
    for(y=0;y<n;y++)
    {
     sum=0;
     pix=0;
     for(i=-p;i<=p;i++)
      for(j=-p;j<=p;j++)
      {
       if(x+i>=0 && x+i<n && y+j>=0 && y+j<n)
       {
        pix++;
        sum+=imagen[x+i][y+j];
       }
      }
     imagenp[x][y]=sum/pix;
    }
  }
```

Ejemplo 28:

Otra operación común en el tratamiento de imágenes es la obtención del histograma de intensidades de los pixels, consistente en hallar, por cada valor posible de la intensidad (0 - 255), el número de pixels que tienen tal valor. Escribir una función histo que reciba como dato una imagen de NxN pixels e imprima en un fichero (histo.dat) el histograma de la imagen según el formato:

Nivel de Gris Número de pixels (una fila por cada nivel de gris)

```
  #define MAX_PIXELS 100
  #define MAX_INTPIX 255
  void histo(int n, int grafo[][MAX_PIXELS])
  {
    int i,j,pix,x,y;
    float sum;
    int histo[MAX_INTPIX];
    FILE *fp;

   for(x=0;x<=MAX_INTPIX;x++)
    histo[x]=0;

   for(x=0;x<n;x++)
    for(y=0;y<n;y++)
     histo[grafo[x][y]]++;
   fp=fopen("histo.dat","w");
   for(y=0;y<MAX_INTPIX;y++)
     fprintf(fp,"%d  %d\n",y, histo[y]);
   fclose(fp);
  }
```

Ejemplo 29:
Función recursiva que que imprime un número decimal en su equivalente en binario.

```
void imprime_binario( unsigned int N )
{
  if( N >= 2 )
      imprime_binario( N >> 1 );    /* todos los demas bits */
  printf( "%d", N & 01 );    /* bit menos significativo */
}
```

Ejemplo 30:
Función recursiva que calcula la suma de los números de 1 a N.

```
/* Calculo de la suma de los primeros N enteros recursivamente */

unsigned long int Suma( unsigned int N )
{
 if( N == 1 )                /* caso base */
   return 1;
 else                        /* llamada recursiva */
   return Suma( N - 1 ) + N;
}
```

Ejemplo 31:
Función recursiva que imprime un número entero positivo como una secuencia de caracteres

```
/* Imprime N como una secuencia de caracteres */

void Imprime_Decimal( unsigned int N )
{
 if( N >= 10 )
   Imprime_Decimal( N / 10 );
 putchar( '0' + N % 10 );
}
```

Ejemplo 32:
Función recursiva que imprime un número entero positivo en cualquier base comprendida entre 2 y 16.

```
/* Imprime N en 2 <= Base <= 16 */

void cambia_base( unsigned int N, unsigned int Base )
{
 static char TablaDigitos[ ] = "0123456789abcdef";

 if( N >= Base )
   cambia_base( N / Base, Base  );
 putchar( TablaDigitos[ N % Base ] );
}
```