# Fundamentos de Programación

Curso Propedéutico
Posgrado en Ingeniería Electrónica
Facultad de Ciencias, UASLP
Junio, 2010

# Descripción y objetivos

- Este curso tiene como objetivos:
  - Repasar los conceptos básicos de programación estructurada en lenguaje C/C++
  - Incrementar la práctica en la elaboración de programas para problemas específicos
  - Evaluar las habilidades de programación de los aspirantes al posgrado

- Este curso NO tiene como objetivos:
  - Enseñar programación “desde cero”
  - Programar en lenguajes distintos al C/C++

# Metodología

- El curso se impartirá en base prácticas en clase, durante las cuales se resolverán las dudas que surjan.

- Al inicio de cada clase se realizará una breve introducción del tema correspondiente; sin embargo, la exposición por parte del profesor será mínima.

- Al final de cada clase, se dejará una tarea la cual debe entregarse a lo mucho tres días después a

fac@fc.uaslp.mx

No se aceptarán tareas retrasadas!

SESION 1

# ESTRUCTURA BASICA DE UN PROGRAMA EN C/C++

# Estructura básica de un programa en C/C++

- Ejemplo de un programa básico en C/C++

```
#include <iostream>

using namespace std;

int main() {
    cout << “Hola mundo!” << endl;
    return 0;
}
```

# Estructura básica de un programa en C/C++

- Ejemplo de un programa básico en C/C++

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hola mundo!" << endl;
    return 0;
}
```

Inclusión de archivos de encabezado

# Estructura básica de un programa en C/C++

- Ejemplo de un programa básico en C/C++

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hola mundo!" << endl;
    return 0;
}
```

Uso global de las librerías estándard

# Estructura básica de un programa en C/C++

- Ejemplo de un programa básico en C/C++

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hola mundo!" << endl;
    return 0;
}
```

Declaración de la función principal

# Estructura básica de un programa en C/C++

- Ejemplo de un programa básico en C/C++

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hola mundo!" << endl;
    return 0;
}
```

# Compilación y ejecución

- Antes de poder ejecutar el programa que escribimos, es necesario compilarlo.

- En DevC++ o CodeBlocks podemos usar los siguientes atajos:
  - F9: Compilar y ejecutar
  - CTRL+F9: Compilar (sin ejecutar)
  - CTRL+F10: Ejecutar (sin recompilar)

# Práctica #1

- Escriba y ejecute un programa que escriba su nombre completo en la pantalla dentro de un marco formado por asteriscos.

- Ejemplo de salida:

```
**************************
*      Me llamo Chamo      *
**************************
```

# Variables y asignación

- Para declarar variables en C/C++ se escribe primero el tipo de variable, seguido del nombre de una o más variables (separados por comas).

- Ejemplos:

```
int a;
int i, j, k;
float x, y;
```

# Asignación de variables

- Para asignar un valor a una variable se utiliza el operador =
- Es posible asignar valores al momento de declarar las variables;
- Ejemplos:

```
x = 10;
float a = 3.4, b = 5.6; c = -1.7;
```

# Tipos de variables

- char – entero de 8 bits
- short – entero con signo de 16 bits
- int – entero con signo de 32 bits
- float – punto flotante de 32 bits
- double – punto flotante de 64 bits

- Modificadores: signed / unsigned / long

# Operadores aritméticos

- En C/C++ se pueden formar expresiones aritméticas con los operadores comunes:
  +, -, *, /

- Debe tenerse siempre en cuenta que los operadores actúan de acuerdo al tipo de datos de los operandos involucrados. Ejemplo:

```
x = 5 / 2;        // el resultado es 2
x = 5.0 / 2;      // el resultado es 2.5
```

- C/C++ no cuenta con un operador de potencia, pero se puede utilizar la función `pow()` de la librería math.h

# Salida de datos a consola

- En C++ se puede usar el flujo `cout` para imprimir información en la pantalla, utilizando el operador de salida de flujo <<

- La constante `endl` imprime un retorno de carro

- Ejemplos:

```
cout << "El valor de x es " << x << endl;
cout << "El area del circulo es " << 3.14159 * r * r;
cout << endl << endl << endl;
```

# Práctica #2

- Desarrolle un programa donde

1. Declare tres variables de punto flotante: a, b, c
2. Declare dos variables de tipo double: x1 y x2
3. Calcule las raíces (reales) x1 y x2 de la ecuación
$$ax^2 + bx + c = 0$$
4. Imprima los datos y resultados en la pantalla

- Ejemplo de salida:

```
Calculo de las raices de una cuadratica
Coeficientes: a = 1, b = -1, c = -2
Raices: x1 = 2, x2 = -1
```

# Entrada de datos desde consola

- El flujo de entrada cin permite introducir datos desde el teclado y almacenarlos en variables.

- Ejemplo:

```
int main() {
   int x;
   cout << "Escribe un numero entero: ";
   cin >> x;
   cout << "El numero que escribiste es: ";
   cout << x << endl;
}
```

# Práctica #3

- Modifique el programa anterior para que el usuario pueda introducir los coeficientes a, b, c, sin necesidad de recompilar el programa.

- El programa deberá imprimir mensajes informativos indicando al usuario qué hace el programa y qué información requiere.

# Tarea #1

1. Escriba un programa que pida al usuario dos enteros y los almacene en las variables A y B. Luego el programa debe intercambiar el valor de A y B.

2. Escriba un programa que calcule e imprima el perímetro y el área de una circunferencia de radio r dado por el usuario.

3. Escriba un programa que, usando solamente dos variables, calcule el promedio de cinco enteros dados por el usuario.

4. Escriba un programa que dados dos puntos en el plano (x1,y1) y (x2,y2), calcule y escriba la distancia entre ellos (sugerencia: utilice la función sqrt() de la librería math.h).

5. Escriba un programa que dados tres puntos en el plano, calcule e imprima las coordenadas del centro y el radio de la circunferencia que pasa por los tres puntos.

SESION 2

# EXPRESIONES BOOLEANAS Y ESTRUCTURAS DE DECISION

# Expresiones booleanas

- Una expresión booleana es aquella que después de evaluarse puede tomar uno de dos posibles resultados: verdadero o falso.
- En lenguaje C/C++, los valores verdadero y falso se representan, respectivamente, con los números 1 y 0.
- Sin embargo, cualquier valor distinto de cero es considerado como “verdadero”.

# Operadores de comparación

- Una manera de formar expresiones booleanas es utilizando los operadores de comparación:

| | |
|---|---|
| a == b | - igualdad |
| a < b | - menor a |
| a > b | - mayor a |
| a != b | - distinto de |
| a <= b | - menor o igual a |
| a >= b | - mayor o igual a |

# Estructuras de decisión

- Una de las principales aplicaciones de las expresiones booleanas es la toma de decisiones:

```
if (expresion) {
  // código a ejecutar si la
  // expresión es verdadera
} else {
  // código a ejecutar si la
  // expresión es falsa
}
```

# Operadores booleanos

- Las expresiones booleanas pueden combinarse mediante los operadores booleanos clásicos:

| | |
|---|---|
| && | - conjunción (and) |
| \|\| | - disyunción (or) |
| ! | - negación (not) |

- Notar que estos operadores difieren de los operadores lógicos binarios: &, |, ~

- Notar que, en muchos casos, los operadores && y || pueden reemplazarse, respectivamente, por el producto y la suma.

# Práctica #1

- Escriba un programa que pida al usuario tres números enteros entre 0 y 10 (inclusive).
- El programa debe verificar que los números están en el rango correcto, y en ese caso imprimir el promedio de los números. En caso contrario debe imprimir un mensaje de error indicando cuál de los números está fuera de rango.

# Práctica #2

- Elabore un programa que pida al usuario los coeficientes de una cuadrática y calcule sus raíces, las cuales posiblemente son complejas. El programa deberá indicar de qué tipo son las raíces (reales, imaginarias o complejas) e imprimir los resultados en el formato adecuado.

# Tarea #2

1. Elabore un programa que pida al usuario cinco números enteros e imprima el menor y el mayor de ellos.

2. Escriba un programa que tome como entrada las coordenadas de un punto en el plano cartesiano e indique al usuario en qué cuadrante se encuentra el punto.

3. Elabore un programa que pida al usuario cuatro calificaciones parciales entre 0 y 10. El programa debe verificar que los datos estén en el rango adecuado y calcular e imprimir el promedio. De acuerdo al promedio, deberá también imprimir alguno de los siguientes mensajes: de 6 a 10: “Aprobado”, de 5 a 6: “Derecho a extraordinario”, de 2 a 5: “Derecho a título”, y de 0 a 2: “Sin derecho a acreditar”.

4. Elabore un programa que pida al usuario su fecha de nacimiento (en el formato día, mes, año), así como la fecha actual, y calcule la edad del usuario en días. El programa debe verificar que las fechas son correctas (e.g., 30/02/1998 y 10/13/1980 no lo son) y tomar en cuenta los años bisiestos.

SESION 3

# INSTRUCCIÓN SWITCH

# Instrucción switch

- La instrucción switch se utiliza cuando se requiere tomar una decisión entre múltiples posibilidades, de acuerdo al valor de una variable o expresión entera.

- Una de las principales aplicaciones es la elaboración de un “menú de opciones” dentro de un programa.

# Sintaxis

```
switch (expresion)  {
    case  c1:        // código a ejecutar en caso de que expresion == c1
                     break;

    case c2:         // código a ejecutar en caso de que expresion == c2
                     break;

    …

    default:         // código (opcional) a ejecutar en caso de que no se cumpla
                     // ninguna de las condiciones anteriores
}
```

* Es importante recordar que c1, c2, etc. deben ser constantes enteras!

# Ejemplo: menú de opciones

Usando switch

```
int a, b, r, opcion;
cout << “1.- Suma” << endl;
cout << “2.- Resta” << endl;
cout << “3.- Producto” << endl;
cout << “4.- División” << endl;
cout << “Elige una opción: “ << endl;
cin << opcion;
switch (opcion) {
      case 1: r = a + b; break;
      case 2: r = a – b; break;
      case 3: r = a * b; break;
      case 4: r = a / b; break;
      default: cout << “Opcion invalida” << endl;
}
cout << “El resultado es “ << r << endl;
```

Usando if

```
int a, b, r, opcion;
cout << “1.- Suma” << endl;
cout << “2.- Resta” << endl;
cout << “3.- Producto” << endl;
cout << “4.- División” << endl;
cout << “Elige una opción: “ << endl;
cin << opcion;
if (opcion == 1) {
      r = a + b;
} else {
      if (opcion == 2) {
            r = a – b;
      } else {
          if (opcion == 3) {
            r = a * b;
          } else {
            if (opcion == 4) {
                  r = a / b;
            } else {
                  cout << “Opcion invalida” << endl;
            }
          }
      }
}
cout << “El resultado es “ << r << endl;
```

# Práctica #1

- En la música occidental, las notas Do, Re, Mi, Fa, Sol, La, y Si suelen representarse mediante las letras C, D, E, F, G, A, y B, respectivamente.

- Elabore un programa que tome como entrada la letra (mayúscula o minúscula) correspondiente a una nota e imprima su nombre. Si la letra dada no corresponde a una nota, el programa debe indicarlo mediante un mensaje de error.

- Sugerencia: utilice el tipo char para almacenar caracteres y recuerde que las constantes de este tipo se escriben entre apóstrofos (e.g., ‘a’, ‘z’, ‘*’, etc.).

# Tarea #3

- Considere el siguiente problema: 3 misioneros y 3 caníbales desean cruzar un río usando una balsa para 2 pasajeros. En ningún momento puede haber mas caníbales que misioneros en cualquier orilla del río. Los movimientos válidos consisten en trasladar uno o dos personajes de una orilla a otra, alternando en cada movimiento el origen y destino.

- Escriba un programa que muestre a los caníbales, misioneros, río, y balsa de alguna manera, y que solicite iterativamente al usuario una de las siguientes opciones:
  - a) Mover un misionero
  - b) Mover un caníbal
  - c) Mover dos misioneros
  - d) Mover dos caníbales

- El programa deberá ajustar el estado del juego de acuerdo a la opción elegida por el usuario, e iterar hasta que el usuario resuelva el problema.

SESION 4

# ESTRUCTURAS DE ITERACION

# Estructuras de iteración

- Las estructuras de iteración se utilizan cuando es necesario repetir un cierto bloque de código un número determinado o indeterminado de veces.

- Todas las estructuras de iteración se basan en la ejecución de un bloque de código de manera repetida mientras el valor de una expresión booleana de control sea verdadero.

- Por lo tanto, es muy importante asegurarse de que la condición se vuelve falsa en algún momento, de manera que el programa no quede atascado en un ciclo infinito.

# Instrucción while

- Sintaxis:

```
while (expresion) {
  // Código a ejecutar de manera repetida
  // mientras la expresión sea verdadera
}
```

# Ejemplo

- El siguiente programa calcula el factorial de *n*, donde *n* es dado por el usuario:

```
int main() {
   int i, n, factorial;
   cout << "Dame el valor de n: ";
   cin >> n;
   factorial = 1;
   i = 1;
   while (i <= n) {
        factorial = factorial * i;
        i = i + 1;
   }
   cout << "El factorial de " << n << " es " << factorial <<
   endl;
}
```

# Práctica #1

- Elabore un programa que pida al usuario un entero positivo *n*. Luego, el programa debe pedir al usuario *n* enteros usando un ciclo. El programa deberá encontrar el mayor, menor, y promedio de los valores dados por el usuario e imprimirlos al final.

# Instrucción do…while

- La instrucción while siempre verifica primero que la condición de control sea verdadera antes de ejecutar el bloque de código.

- Si se desea ejecutar el bloque de código al menos una vez antes de verificar la condición, se puede usar la instrucción do…while:

```
do {
  // bloque de código a ejecutar al menos una vez
} while (expresion);
```

# Práctica #2

- Realice un programa que pida al usuario una serie de números enteros no negativos utilizando un ciclo do…while. Una vez que el usuario introduzca un número negativo, el ciclo deberá terminar y el programa deberá imprimir el mayor, menor, y promedio de todos los números ingresados (excepto el último).

# Tarea #4

1. Elabore un programa que implemente el método de bisección para encontrar la raíz en el intervalo (0, 3) de la función

$$f(x) = \log(1+\cos x) - \frac{\sqrt{x}}{2}$$

2. Elabore un programa que implemente la regla del trapecio para calcular la integral

$$\int_0^{2\pi} \frac{1}{1+\sqrt{x}} e^{(\sin x)/(x+1)^2} dx$$

3. Elabore un programa que elija un número aleatorio n entre 1 y 100 (usar funciones rand() y srand() de stdlib.h). El programa dará al usuario 5 oportunidades para adivinar el número: en cada oportunidad, el usuario ingresará su respuesta y el programa indicará si la respuesta es menor, mayor, o igual a n (en cuyo caso el usuario habrá ganado). Si después de 5 intentos el usuario no ha adivinado, entonces habrá perdido el juego.

# Tarea #4 (sugerencias)

- **Algoritmo de bisección:**

  Dado *el intervalo (a, b)* y una función *f(x)* contínua en *(a,b)*:

  1. Verificar que *f(a) f(b) < 0.* De lo contrario, no podemos asegurar que existe una raíz en el intervalo dado y el algoritmo debe terminar.
  2. Hacer *r=a*. Iterar hasta que *r* converja:
     a. Obtener una aproximación de la raíz *r como r = (a+b) /* 2 .
     b. Si *f(a) f(r) < 0,* entonces la raíz está en la mitad izquierda del intervalo. Por lo tanto, hacer *b = r.*
     c. En caso contrario, entonces la raíz está en la mitad derecha del intervalo. Por lo tanto, hacer *a = r.*

- **Regla del trapecio:**

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[ f(a) + f(b) + 2\sum_{k=1}^{n-1} f(a+kh) \right],$$

donde *n* es el número de segmentos y *h = (b-a)/n*.

SESION 5

# ESTRUCTURAS DE ITERACION II

# La instrucción for

- La otra estructura en C/C++ para la realización de ciclos es proporcionada por la instrucción for, cuya sintaxis es:

```
for (inicializacion;  condicion;  incremento) {
  // bloque de instrucciones
}
```

Donde `inicializacion` es una expresión que se ejecuta antes de iniciar el ciclo. El ciclo se ejecuta mientras la `condicion` sea verdadera, y en cada iteración (despues de ejecutar el bloque de instrucciones) se evalua la expresion `incremento`.

# Equivalencia entre for y while

- Las siguientes estructuras son equivalentes:

```
for (inicializacion;  condicion;  incremento) {
      // bloque de instrucciones
}
```

```
inicializacion;
while (condicion) {
      // bloque de instrucciones
      incremento;
}
```

# Ejemplo

- La instrucción for se usa típicamente para realizar ciclos controlados por contador, ya que permite ver todos los parámetros del ciclo en una sola línea:

```
int main() {
   int i, n, f = 1;
   cout << "Dame un numero entero: ";
   cin >> n;
   for (i = 1; i <= n; i++) {
       f = f * i;
   }
   cout << n << "! = " << f << endl;
}
```

# Ciclos anidados

- En muchas aplicaciones se requiere utilizar dos o más ciclos anidados, de manera que un ciclo exterior no complete una iteración hasta que los ciclos interiores hayan finalizado.

- Una manera de entender el funcionamiento de los ciclos anidados es comparándolos con el odómetro de un coche:

# Ejemplo

```
#include <iostream>
#include <cstdlib>
#include <time.h>

using namespace std;

int main() {
    int i, j, k;
    int t0;
    for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
            for (k = 0; k < 10; k++) {
                cout << "\r" << i << j << k;
                cout.flush();
                t0 = time(0);
                while (t0 == time(0)) {}
            }
        }
    }
    return 0;
}
```

# Práctica #1

a) Escriba un programa que, dado un número entero ***n***, determine si ***n*** es primo o no. Sugerencia: verifique si ***n*** es divisible entre algún entero entre 2 y sqrt(***n***).
b) Modifique el programa anterior para que pida al usuario un entero positivo ***N***, y luego imprima los primeros ***N*** números primos.

Ejemplo: si el usuario ingresa 10, el programa debe imprimir: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

# Tarea #5

1. Elabore un programa que utilice uno o mas ciclos `for` para factorizar un número entero ***n*** (dado por el usuario) como el producto de números primos. Sugerencia: pruebe si ***n*** es divisible entre 2, 3, …, ***n***; cuando lo sea, divida ***n*** entre el factor y repita la operación hasta que ***n*** sea 1.

2. Se define la función ***u(n)*** para n entero no negativo como

$$u(n) = \begin{cases} a, & \text{si } n = 0, \\ u(n-1)/2, & \text{si } n > 0 \text{ y } u(n-1) \text{ es par}, \\ 3u(n-1)+1, & \text{si } n > 0 \text{ y } u(n-1) \text{ es impar}. \end{cases}$$

   donde ***a*** es un entero, y puede mostrarse que para cualquier valor de ***a*** existe un entero $N_a$ tal que $u(N_a) = 1$.

   Elabore un programa que pida al usuario el valor de ***a***, y luego calcule e imprima todos los valores de ***u(n)*** desde ***n = 1*** hasta $n = N_a$.

3. Elabore un programa que calcule e imprima el volumen bajo la superficie $f(x,y) = x^2 + y^2 - r^2$, donde $r^2$ está dado por el usuario, en el intervalo $|x|<1, |y|<1$. Para esto, divida el intervalo en una rejilla de ***n*** x ***n*** (donde ***n*** es dado por el usuario) y utilice integración por paralelepípedos rectangulares.

4. Modifique el programa anterior para que imprima una tabla de la integral de ***f(x,y)*** para valores enteros de ***r*** desde 1 hasta 20.

SESION 6

# FUNCIONES

# Funciones

- Una función es una sección de código que realiza una tarea específica, y que puede ser llamada desde otra función.

- Todo programa en C/C++ se compone de una o más funciones, incluyendo la función main().

- El lenguaje C incluye muchas funciones de utilidad general, las cuales están organizadas en librerías.

# Definición de funciones

- Una función en C/C++ se define de la manera siguiente:

```
tipo  nombre  (argumento1, argumento2, … ) {
  // instrucciones a realizar
}
```

donde `tipo` representa el tipo de datos del valor devuelto por la función, y los argumentos son variables (con especificación de tipo) donde se guardan los parámetros de la función.

# Ejemplo

```
int factorial(int n) {
   int i, f = 1;
   for (i = 1; i <= n; i++) {
       f = f * i;
   }
   return f;
}

int main() {
   int n;
   for (n = 1; n <= 10; n++) {
       cout << n << "! = " << factorial(n) << endl;
   }
}
```

# Funciones tipo void

- Es posible definir una función que no devuelva ningún resultado si se declara de tipo void:

```
void Retardo(int segundos) {
  int t0 = time();
  while (time() - t0 < segundos) {}
}
```

Notar que, en este caso, no es necesario usar la instrucción return a menos que se desee terminar la función de manera prematura.

# Práctica #1

a) Implemente una función f(x) que tome por parámetro un número *x* (de punto flotante) y devuelva $3\cos(x) - 1$

b) Implemente una función llamada bisección que tome por parámetros *a* y *b*, y encuentre la raíz de la función f(*x*) en el intervalo [*a,b*]. En caso de no existir una raíz, la función deberá devolver el valor de *a*.

Ninguna de las dos funciones anteriores debe pedir datos al usuario o imprimir información en la pantalla.

c) Dentro de la función main(), pida al usuario los valores de *a* y *b*, y llame a la función de bisección para encontrar e imprimir la raíz de f(*x*) en [*a,b*]. En caso de no existir una raíz, el programa deberá actuar de manera acorde.

# Paso de argumentos por valor

- Por lo general, las funciones reciben argumentos por valor. Esto significa que los parámetros son variables que solo existen dentro del contexto de la función:

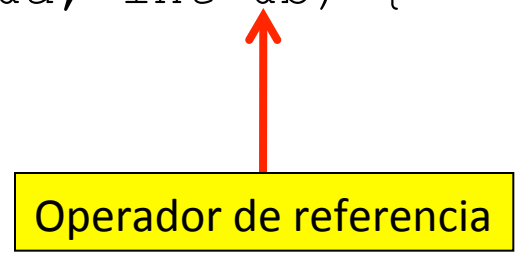
```
void intercambia(int a, int b) {
   int temp = a;
   a = b;
   b = temp;
}

int main() {
   int a = 1, b = 2;
   intercambia(a, b);
   cout << "a = " << a << ", b = " << b << endl;
}
```

- La salida de este programa es: `a = 1, b = 2`

# Paso de argumentos por referencia

- El paso por referencia asocia los parámetros de una función con variables que existen fuera del contexto de la función, lo cual permite modificar su valor:

```
void intercambia(int &a, int &b) {
   int temp = a;
   a = b;
   b = temp;
}
```




- El paso por referencia es útil cuando se requiere que una función devuelva más de un resultado.

# Práctica #2

a) Escriba su propia versión de la función exp(x) utilizando la siguiente serie:

$$e^x \approx \sum_{n=0}^{N} \frac{x^n}{n!}$$

donde N es dado como segundo argumento.

b) Utilice la función anterior para escribir una función que calcule la densidad de probabilidad de una distribución Gaussiana con media $\mu$ y varianza $\sigma^2$, dada por:

$$n_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left( -\frac{(x-\mu)^2}{2\sigma^2} \right)$$

c) Utilice la regla del trapecio para mostrar que $\int_{-4\sigma}^{4\sigma} n_{\mu,\sigma}(x)\, dx \approx 1.$

# Tarea #6

1. Escriba una función llamada `car2pol` que tome como argumentos las coordenadas de un punto ***(x,y)*** en el plano cartesiano, así como las coordenadas polares ***(r,α)*** por referencia. La función deberá calcular ***r*** y ***α*** a partir de ***x*** y ***y***. De manera similar, escriba una función `pol2car(r,a,x,y)` que realice la operación inversa. Para probar ambas funciones, pida al usuario (dentro de la función `main`) las coordenadas cartesianas de un punto, conviértalas a polares y nuevamente a cartesianas, e imprima el resultado.

2. Escriba funciones llamadas `mcd(a,b)` y `mcm(a,b)` que calculen y devuelvan, respectivamente, el máximo común divisor y el mínimo común múltiplo de sus dos argumentos enteros. Use estas funciones dentro de la función `main()` para imprimir el mcd y mcm de dos números dados por el usuario.

3. Escriba una función que calcule el coeficiente binomial:

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!},$$

tratando de que funcione para el mayor valor de *n* posible. Escriba una función main para probar la función anterior.

SESION 7

# PROGRAMACION ESTRUCTURADA

# Programación estructurada

- Un programa estructurado se compone de tres tipos básicos de estructuras de flujo:

1. Secuencia: la cual indica el orden de ejecución de las instrucciones.

2. Selección: mediante la cual una secuencia de instrucciones se ejecuta dependiendo del estado del programa.

3. Repetición: mediante la cual una secuencia de instrucciones se ejecuta de manera iterativa mientras el programa se encuentre en un cierto estado.

# Programación procedural

- La programación procedural es un paradigma derivado de la programación estructurada en el cual un programa se divide en procedimientos, subrutinas, o funciones, los cuales simplemente consisten de una serie de instrucciones a ser ejecutadas.

- Cualquier procedimiento puede ser llamado desde cualquier punto del programa, incluyendo otros procedimientos, o el mismo.

# Modularidad y alcance

- Dentro de la programación procedural, las siguientes características son deseables:
  - Modularidad: Para cada procedimiento se especifica un conjunto de argumentos de entrada y valores de salida que le permiten comunicarse con otros procedimientos.
  - Alcance: Cada procedimiento cuenta con su propio espacio de variables, las cuales no pueden ser accedidas desde otros procedimientos, salvo mediante los mecanismos de paso de argumentos y retorno de valores.

# Ventajas de la programación procedural

- Diseño descendente: La modularidad permite al programador plantear primero una solución al problema completo, para luego enfocarse en los detalles.

- Cooperación: Múltiples programadores pueden trabajar en un programa complejo, cada uno desarrollando un módulo distinto.

- Reusabilidad: Los procedimientos escritos para un programa pueden reutilizarse en otros programas  que requieran la misma tarea.

- Fácil depuración: Ya que cada procedimiento realiza una tarea especializada, es posible depurar cada procedimiento de manera individual.

- Fácil mantenimiento: Un programa procedural que se escribe correctamente es fácil de entender, mantener y actualizar, incluso por otros programadores.

# Diseño descendente (top-down)

- En programación, el diseño descendente consiste en dividir una tarea compleja en sub-tareas, y resolver cada sub-tarea de manera independiente hasta que cada paso pueda implementarse fácilmente.

# Ejemplo de diseño descendente

- Considere el problema de elaborar un sistema de navegación para un robot autónomo.
- El sistema puede dividirse en las siguientes etapas:
  - Capturar información sobre el entorno a través de sensores (cámaras, radares, micrófonos, etc).
  - Procesar la información del entorno para extraer rasgos relevantes (obstáculos, objetos en movimiento, corredores, etc.)
  - Tomar decisiones de acuerdo a los rasgos observados y el estado actual (acelerar, detener, girar, etc.)
  - Actualizar el estado del sistema de acuerdo a las decisiones tomadas (ajustar velocidad, dirección, etc.)
- Cada una de estas etapas puede dividirse a su vez en otras sub-etapas.

# Alcance de variables y funciones

- El alcance de una variable o función es el contexto desde el cual puede accederse a la misma.

- En caso de ambigüedad (variables con mismo nombre pero distinto alcance), el lenguaje C/C ++ elegirá la variable declarada en el contexto mas cercano.

# Ejemplo de alcance

```
int x = 1;

int f() {
   return x;
}

int main() {
   int x = 2;
   {
      int x = 3;
      cout << x << endl;
   }
   cout << x << endl;
   cout << f() << endl;
}
```

¿Qué imprime este programa?

# Tarea #7

- De los dos programas siguientes, desarrolle el que le haya sido asignado, aplicando los conceptos de programación estructurada y diseño descendente.

- El programa deberá cumplir con lo siguiente:
  - La función main() solo debe contener declaraciones de variables y llamadas a otras funciones.
  - El programa debe dividirse en funciones, cada una de las cuales debe realizar una tarea específica.
  - No debe haber variables globales. Las funciones compartirán información mediante los mecanismos de paso de argumentos y retorno de valores.

# Tarea #7 – Programa 1

- Elabore un programa para medir la velocidad promedio de cálculo aritmético de una persona. El programa deberá presentar al usuario una serie de operaciones (suma, resta, multiplicación y división) elegidas al azar. Para el caso de suma y resta, los operandos serán de máximo tres dígitos, para la multiplicación de máximo dos dígitos, y para la división de tres dígitos para el dividendo y uno para el divisor, y el resultado siempre debe ser entero.

- Al iniciar, el programa pedirá al usuario el número de operaciones a realizar. Posteriormente mostrará la operación actual (e.g., “182 + 324 = ”) y esperará a que el usuario ingrese la respuesta. El programa llevará la cuenta del número de respuestas correctas, así como del tiempo de respuesta promedio (calculado solo para las respuestas correctas). Al final, el programa reportará estos resultados.

- Subrutinas a considerar:  imprimir pantalla o menú inicial, pedir datos iniciales al usuario, elegir operación, ejecutar prueba, reportar resultados.

# Tarea #7 – Programa 2

- Elabore un programa que implemente el juego de 3-en-raya (Gato) para dos jugadores humanos. El programa deberá inicialmente mostrar el tablero con las casillas numeradas del 1 al 9, y, de manera iterativa, pedir la casilla donde el siguiente jugador desea colocar su ficha, verificando que el movimiento sea válido. También deberá determinar en qué momento hay un ganador, o si llega a haber un empate.

- Sugerencia: utilice nueve variables tipo char para representar cada una de las casillas del tablero. Cada casilla puede contener un espacio vacío, una 'X', o una 'O'.

- Subrutinas a considerar: solicitar jugador inicial (ficha X/O), dibujar tablero, solicitar siguiente movimiento, determinar ganador.

SESION 8

# ARREGLOS

# Motivación

- La mayoría de las aplicaciones actuales requieren el manejo de grandes cantidades de datos.

- Si requerimos almacenar N datos, uno podría declarar variables llamadas x1, x2, …, xN. Sin embargo, esto suele ser impráctico, ya que no es posible generalizar ciertas tareas para que se apliquen a todas las variables.

- Considere, por ejemplo, el problema de estimar el histograma de serie de tiros de dado: en cada tiro el valor obtenido está entre 1 y 6. Después de N tiros, deseamos saber cuántas veces cayó 1, 2, 3, etc.

# Histograma: solución ingenua

```
int main() {
    int h1, h2, h3, h4, h5, h6;
    int i, x, N = 1000;
    h1 = h2 = h3 = h4 = h5 = h6 = 0;
    for (i = 0; i < N; i++) {
          x = rand() % 6 + 1;
          switch (x) {
                    case 1: h1++; break;
                    case 2: h2++; break;
                    case 3: h3++; break;
                    case 4: h4++; break;
                    case 5: h5++; break;
                    case 6: h6++; break;
          }
    }

    cout << "Tabla de frecuencias:" << endl;
    cout << "h(1) = " << h1 << endl;
    cout << "h(2) = " << h1 << endl;
    cout << "h(3) = " << h1 << endl;
    cout << "h(4) = " << h1 << endl;
    cout << "h(5) = " << h1 << endl;
    cout << "h(6) = " << h1 << endl;
    return 0;
}
```

# Arreglos

- Una mejor solución consiste en declarar un *arreglo*, el cual consiste en un grupo de variables del mismo tipo y con el mismo nombre, pero que se distinguen mediante un *subíndice* entero.

- La forma de declarar un arreglo en C/C++ es:

```
tipo_datos  nombre_arreglo[numero_elementos];
```

# Arreglos

- En C/C++, los elementos de un arreglo de tamaño N están indexados a partir de cero hasta N-1.

- Para acceder a los elementos individuales de un arreglo, se escribe el nombre del arreglo seguido de el subíndice del elemento entre corchetes.

- Ejemplo:

```
int a[3];
a[0] = 5;
a[1] = 8;
a[2] = a[0] * a[1];
```

# Histograma: solución con arreglos

```
int main() {
   int h[6];
   int i, x, N = 1000;
   for (i = 0; i < 6; i++) { h[i] = 0; }
   for (i = 0; i < N; i++) {
       x = rand() % 6 + 1;
       h[x - 1]++;
   }

   cout << "Tabla de frecuencias" << endl;
   for (i = 0; i < 6; i++) {
       cout << "h(" << i+1 << ") = " << h[i] << endl;
   }
   return 0;
}
```

# Precaución!

- El compilador de C/C++ no verifica que los subíndices de los elementos de un arreglo estén dentro de los límites correspondientes.

- El programador debe tener cuidado de no sobrepasar estos límites. De lo contrario, es posible que sobreescriba la memoria asignada a otras variables. Ejemplo:

```
int main() {
    int a[10], x[10];
    a[0] = 1;
    x[12] = 2;
    cout << a[0] << endl;
}
```

# Arreglos como argumentos de funciones

- Es posible pasar un arreglo como argumento de una función.

- En la mayoría de los casos, es necesario también pasar como argumento el tamaño del arreglo.

- Ejemplo:

```
float promedio(float x[], int n);
```

# Histograma: solución general

```
void histo(float x[], int nx, int h[], float a, float b, int nbins) {
   int i, j;
   float binsize = (b – a) / nbins;
   for (i = 0; i < nbins; i++) { h[i] = 0; }
   for (i = 0; i < nx; i++) {
        j = (int)((x[i] – a) / binsize);
        if ((j >= 0) && (j < nbins)) h[j]++;
   }
}

int main() {
   const int N = 1000;
   float x[N];
   int i, h[6];
   for (i = 0; i < N; i++) { x[i] = rand() % 6 + 1; }
   histo(x, N, h, 1, 6.1, 6);
   for (i = 0; i < 6; i++) { cout << "h(" << i << ") = " << h[i] << endl;
   return 0;
}
```

# Práctica #1

1. Escriba una función llamada minmax() que tome como parámetros un arreglo de floats y su tamaño, así como cuatro variables por referencia: min, nmin, max, nmax. La función deberá encontrar el valor mínimo y máximo en el arreglo y guardarlos en min y max, respectivamente, así como los subíndices correspondientes en nmin y nmax.

2. Utilizando la función anterior, escriba otra función para normalizar los datos de un arreglo al rango [0,1] mediante la siguiente transformación:

$$x \leftarrow \frac{x - \min}{\max - \min}.$$

3. Escriba un programa para probar las dos funciones anteriores, utilizando como datos de ejemplo un arreglo de números aleatorios entre -10 y 10.

# Tarea #8

1. Escriba una función que tome tres arreglos como entrada: *x*[], *h*[], *y*[], junto con sus respectivos tamaños $n_x$, $n_h$, $n_y$. La función debe calcular la convolución de *x* y *h*, definida como:

$$y[n] = \sum_{k=0}^{n_x - 1} x[k]\, h[n-k],$$

asumiendo que h[] vale cero fuera de sus límites. Escriba un programa de prueba que calcule la convolución de las señales [3, -1, 2, 4] y [-3, 1, 3, -1], considerando $n_y = n_x + n_h - 1$. Verifique la conmutatividad de la convolución.

2. Escriba una función que tome como parámetros un arreglo de enteros y su tamaño. La función debe ordenar el arreglo de manera ascendente. Luego escriba otra función que utilice la anterior para encontrar la mediana de un arreglo de enteros. Escriba un programa para probar ambas funciones.

3. La entropía de un arreglo *X*[] se puede definir como: $H(X) = -\sum_{i=0}^{n-1} p_i \log p_i$,

donde $p_i = h_i / N$, $h_i$ es el *i*-ésimo bin de un histograma de *X*, *n* es el número de bins, y *N* es el tamaño de *X*. Escriba una función que tome como argumentos un arreglo, su tamaño, y el número de bins, y devuelva la entropía del arreglo. Utilice las funciones histo() y minmax() vistas en clase. Escriba un programa de prueba.

SESION 9

# ARREGLOS MULTIDIMENSIONALES

# Motivación

- En muchas aplicaciones, se requiere organizar un gran número de datos en un arreglo cuyos elementos se distingan por medio de dos o más subíndices.

- Ejemplos:
  - Matrices (subíndices: renglón, columna)
  - Imágenes (subíndices: posición (x,y) de un pixel)
  - Video (subíndices: posición (x,y) y tiempo t)
  - Señales electrofisiológicas (EEG, EKG, fMRI)

# Arreglos bidimensionales

- Supongamos que nuestra aplicación requiere de arreglos organizados en forma de matriz, de manera que se haga referencia a cada elemento mediante dos subíndices.

- En C/C++ existen dos técnicas para implementar este tipo de arreglos:

  1. Arreglos de arreglos
  2. Orden lexicográfico

# Arreglos de arreglos

int x[M][N];

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| x[0] | x[0][0] | x[0][1] | | | | | | | | x[0] [N-1] |
| x[1] | x[1][0] | x[1][1] | | | | | | | | x[1] [N-1] |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| x[M-1] | x[M-1] [0] | x[M-1] [1] | | | | | | | | x[M-1] [N-1] |

# Ejemplo: arreglos de arreglos

- Suponga que se desea implementar un arreglo de M x N donde el elemento (i,j) sea igual a (i + j).

- Solución #1:

```
int i, j;
int x[M][N];
for (i = 0; i < M; i++) {
   for (j = 0; j < N; j++) {
       x[i][j] = i + j;
   }
}
```

# Orden lexicográfico

int x[M*N];

x

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | | | N-1 |
| N+0 | N+1 | | | | | | | | N+(N-1) |
| 2N+0 | 2N+1 | | | | | | | | 2N+ (N+1) |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| (M-1)N+ 0 | (M-1)N+ 1 | | | | | | | | (M-1)N+ (N-1) |

# Ejemplo: orden lexicográfico

```
int i, j;
  int x[M*N];
  for (i = 0; i < M; i++) {
     for (j = 0; j < N; j++) {
         x[i*N+j] = i + j;
     }
  }
```

# Arreglos bidimensionales como argumentos

## Arreglos de arreglos

```
void muestra(int x[][10], int m) {
    int i, j;
    for (i = 0; i < m; i++) {
        for (j = 0; j < 10; j++) {
            cout << x[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
   int x[10][10];
   int i, j;
   for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
                x[i][j] = i + j;
        }
   }
   imprime(x, 10);
}
```

## Orden lexicográfico

```
void muestra(int x[], int m, int n) {
    int i, j;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            cout << x[i*n+j] << " ";
        }
        cout << endl;
    }
}

int main() {
   int x[10*10];
   int i, j;
   for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
                x[i*10+j] = i + j;
        }
   }
   imprime(x, 10, 10);
}
```

# Comparación entre técnicas

| | Arreglos de arreglos | Orden lexicográfico |
|---|---|---|
| Ventajas | -Fácil acceso a elementos<br><br>-Fácil acceso a renglones<br><br>-Fácil extensión al caso multidimensional | -Pueden ser muy eficientes si se programa de manera adecuada.<br><br>-Es posible pasar arreglos de dimensión y tamaño arbitrarios como argumentos de una función. |
| Desventajas | -Uso limitado como argumentos de una función: solo para arreglos cuyas dimensiones (excepto la primera) están fijas. | -El acceso a elementos requiere mayor cuidado y el cálculo de la posición del elemento dentro del arreglo lineal. |

# Arreglos multidimensionales

- También es posible utilizar arreglos de más de dos dimensiones; es decir, arreglos en los cuales los elementos estén indexados mediante mas de dos subíndices.

- Ejemplo: arreglos tridimensionales

```
// Técnica: arreglo de arreglos
// Declaración
int x[M][N][P];

// Acceso al elemento (i,j,k)
x[i][j][k] = 0;
```

```
// Técnica: orden lexicográfico
// Declaración
int x[M*N*P];

// Acceso al elemento (i,j,k)
x[i*N*P + j*P + k] = 0;
```

# Práctica #1

- Las operaciones elementales de una matriz son:
  - Multiplicar el renglón *i* por una constante *k*.
  - Sumar al renglón *i* el renglón *j* multiplicado por *k*.
  - Intercambiar los renglones *i* y *j*.

- Escriba tres funciones, una para cada operación elemental. Las funciones deben tomar como argumentos la matriz, sus dimensiones, y los parámetros requeridos para la operación.

# Práctica #2

- Utilizando las funciones anteriores, implemente en una función el método de Gauss-Jordan para encontrar la solución del sistema $Ax = b$. La función debe tomar $A$, $b$, y el tamaño del sistema $n$ como argumentos:

1. Iniciar con $c = 0$ y $r = 0$.
2. Mientras $c < n$ y $r < n$, hacer
   a. Sea $j = \arg\max_{i = r,\ldots,n-1} \{A(i,c)\}$
   b. Si $r$ es distinto de $j$, intercambiar renglones $r$ y $j$.
   c. Multiplicar el renglón $r$ por $1/A(r,c)$
   d. Si $A(r,c)$ es distinto de cero
      i. Para cada renglón $i = r + 1, \ldots, n$, sumar al renglón i el resultado de multiplicar el renglón $r$ por $-A(i,c)$.
      ii. Incrementar $r$.
   e. Incrementar $c$.
3. Para $r$ desde $n$-1 hasta 0,
   1. Para cada renglón $i = 0,\ldots,r$-1, sumar al renglón $i$ el resultado de multiplicar el renglón $r$ por $-A(i,r)$.

# Tarea #9 – Parte 1

1. Elabore una función que realice el producto *C = AB* de dos matrices *A* de *m* x *n* y *B* de *n* x *p*. La función debe tomar como parámetros *A, B, C, m, n* y *p*.

2. Elabore una función que, dado el arreglo *x[]* de tamaño *N*, calcule el arreglo *X[k]* para *k = 0,…,N-1*, dado por:

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\{-i(2\pi kn / N)\}.$$

*X[k]* se conoce como la transformada discreta de Fourier (TDF) de *x[n]*, y es una señal compleja la cual puede representarse como un arreglo de *N* x 2. Dada *X[k]*, es posible recuperar *x[n]* mediante la transformada discreta inversa de Fourier (TDIF):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \exp\{i(2\pi kn / N)\}.$$

Elabore una función que implemente la (TDIF) y verifique que ambas funciones son una inversa de la otra calculando el error cuadrático medio entre una señal de ruido aleatorio *x[n]*, y la versión recuperada a partir de la TDIF de la TDF de *x[n]*.

# Tarea #9 – Parte 2

3. Considere dos variables aleatorias *X* y *Y*, que pueden tomar valores entre 0 y *K-1*. El histograma conjunto $h_{ij}$ de *X* y *Y* es una matriz que representa el número de veces que se observa simultáneamente *X=i* y *Y=j* en una muestra. Escriba una función que tome como argumentos los arreglos *X* y *Y*, ambos de tamaño *N* y con valores entre 0 y *K-1*, y calcule el histograma conjunto (de tamaño *K* x *K*).

4. La información mutua de dos variables aleatorias *X* y *Y* es una medida de qué tan fácil es predecir *X* cuando se conoce *Y*, y se calcula como

$$I(X,Y) = H(X) + H(Y) - H(X,Y),$$

donde *H(X)* es la entropía de *X*, y *H(X,Y)* es la entropía conjunta de *X* y *Y*, la cual puede calcularse a partir del histograma conjunto de *X* y *Y* como:

$$H(X,Y) = -\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} p_{ij} \log p_{ij},$$

donde $p_{ij} = h_{ij} / N$. Escriba una función que calcule la información mutua de dos arreglos.

SESION 10

# APUNTADORES Y ASIGNACION DINAMICA DE MEMORIA

# Almacenamiento en memoria

- La memoria puede verse como un conjunto de celdas numeradas y ordenadas, cada una de las cuales puede almacenar un byte de información. El número correspondiente a cada celda se conoce como su *dirección*.

- Cuando se crea una variable, el compilador reserva el número suficiente de celdas de memoria (bytes) requeridos para almacenar la variable, y se encarga de que los espacios reservados no se traslapen.

# Direcciones de memoria

- En C/C++ es posible obtener la dirección de memoria donde se encuentra almacenada una variable mediante el operador de referencia &

- Ejemplo:

```
int main() {
   int a = 10, b = 20;
   cout << &a << endl;
   cout << &b << endl;
   return 0;
}
```

# Tamaño de una variable

- Podemos también obtener la cantidad de memoria que ocupa una variable (en bytes) mediante el operador sizeof:

```
int main() {
    int a = 10;
    cout << "Valor de a: " << a << endl;
    cout << "Dirección de a: " << &a << endl;
    cout << "Tamaño de a: " << sizeof(a) << endl;
    return 0;
}
```

# Apuntadores

- Un apuntador es una variable que contiene una dirección de memoria (donde posiblemente se almacene el valor de otra variable).

- Para crear apuntadores, se utiliza el operador * como se muestra a continuación:

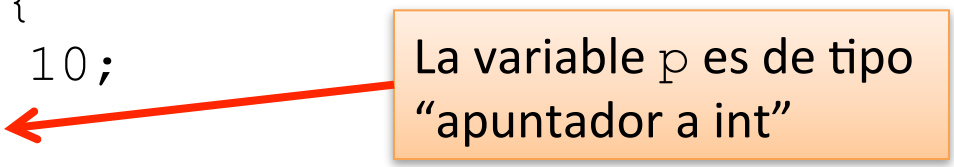
```
int main() {
    int a = 10;
    int *p;
    p = &a;
    cout << "Valor de p: " << p << endl;
    cout << "Valor en p: " << *p << endl;
    cout << "Dirección de p: " << &p << endl;
}
```

# Apuntadores

- Un apuntador es una variable que contiene una dirección de memoria (donde posiblemente se almacene el valor de otra variable).

- Para crear apuntadores, se utiliza el operador * como se muestra a continuación:

```
int main() {
    int a = 10;
    int *p;
    p = &a;
    cout << "Valor de p: " << p << endl;
    cout << "Valor en p: " << *p << endl;
    cout << "Dirección de p: " << &p << endl;
}
```



# Almacenamiento de arreglos

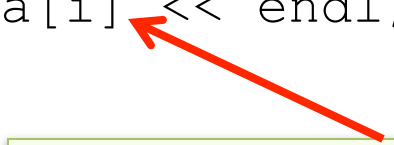
- Cuando se declara un arreglo, se reserva un solo bloque contiguo de memoria para almacenar todos sus elementos, y éstos se almacenan en la memoria en el mismo orden que ocupan en el arreglo:

```
int main() {
    int i, a[10];
    for (i = 0; i < 10; i++) {
          cout << &a[i] << endl;
    }
    return 0;
}
```

# Almacenamiento de arreglos

- Cuando se declara un arreglo, se reserva un solo bloque contiguo de memoria para almacenar todos sus elementos, y éstos se almacenan en la memoria en el mismo orden que ocupan en el arreglo:

```
int main() {
    int i, a[10];
    for (i = 0; i < 10; i++) {
        cout << &a[i] << endl;
    }
    return 0;
}
```



Verificar que `&a[i]` es igual a `a[0] + i * sizeof(int)`

# Arreglos y apuntadores

- En C/C++, el nombre de un arreglo es también un apuntador al primer elemento del arreglo.

- De hecho, el lenguaje C no puede distinguir entre un arreglo y un apuntador: ambos son completamente intercambiables.

```
int main() {
    int i, a[10], *p;

    for (i = 0; i < 10; i++) { a[i] = i; };
    p = a;
    for (i = 0; i < 10; i++) {
        cout << p[i] << endl;
    }

    return 0;
}
```

# Aritmética de apuntadores

- Recuerde que un apuntador siempre está asociado con el tipo de dato al que apunta (e.g., apuntador a int, apuntador a float, etc).

- Esto hace posible definir la operación `p+k` donde `p` es un apuntador y `k` es un entero. El resultado de esta operación es

```
p + k * sizeof(tipo)
```

  donde tipo es el tipo de datos asociado a `p`.

- Notar entonces que, si `p` es un arreglo

  `&p[k]` es equivalente a `p+k` y `p[k]` equivale a `*(p+k)`

# Ejemplo: operaciones elementales

```
void MulRen(float *m, int n, int r, float k) {
   float *p = m + r * n;
   for (int j = 0; j < n; j++) { p[j] *= k; }
}

void SumRen(float *m, int n, int r1, int r2, float k) {
   float *p1 = m + r1 * n;
   float *p2 = m + r2 * n;
   for (int j = 0; j < n; j++) { p1[j] += p2[j] * k; }
}

void InterRen(float *m, int n, int r1, int r2) {
   float t;
   float *p1 = m + r1 * n;
   float *p2 = m + r2 * n;
   for (int j = 0; j < n; j++) {
       t = p1[j];  p1[j] = p2[j];  p2[j] = t;
   }
}
```

# Asignación dinámica de memoria

- Para cada función, el compilador de C/C++ reserva una cierta cantidad de memoria para almacenar las variables locales.

- En algunas ocasiones será necesario utilizar un arreglo que requiera una mayor cantidad de memoria:

```
int main() {
   int arreglote[1024 * 1024];
   return 0;
}
```

# Asignación dinámica de memoria

- La solución consiste en asignar memoria a un arreglo de manera dinámica mediante el operador `new`:

```
int main() {
    int i, n = 1024 * 1024;
    int *a = new int[n];
    if (a != NULL) {
       for (i = 0; i < n; i++) { a[i] = i; }
       delete[] a;
    }
    return 0;
}
```

- Es importante siempre liberar la memoria reservada mediante el operador `delete[]` una vez que ya no se utiliza.

# El apuntador NULL

- El lenguage C/C++ define una constante especial llamada NULL, el cual es un apuntador que no apunta a ningún lugar válido en la memoria y tiene valor numérico cero.

- Se recomienda siempre inicializar los apuntadores con una dirección válida, o bien, con NULL, de manera que no sea posible sobreescribir información accidentalmente.

- El operador `new` devuelve NULL si no es capaz de reservar la cantidad de memoria solicitada. Esto permite detectar la falta de memoria en un programa.

# Práctica #1

- Declare un arreglo bidimensional `X` de `M` x `N` elementos como un “arreglo de arreglos” y llénelo con números aleatorios entre 0 y 10.

- Imprima las direcciones de memoria de cada uno de los elementos y verifique que éstos están organizados en orden lexicográfico.

- Declare un apuntador `p` y asígnele la dirección del primer elemento de `X`. Ahora `p` actúa como un arreglo unidimensional que hace referencia a los mismos elementos que `X`, pero en orden lexicográfico. Utilice `p` para imprimir el contenido del arreglo.

# Tarea #10 – Parte 1

- Escriba una función llamada `promedio(x,n)` que calcule y devuelva el promedio de un arreglo *x* de tamaño *n*. Escriba otra función llamada `punto(x,y,n)` que devuelva el producto punto de dos vectores *x* y *y*, ambos de tamaño *n*.

- Considere el problema de regresión lineal simple, donde uno desea encontrar una función de la forma $y(x) = mx + b$, dado un conjunto de *n* datos $(x_i, y_i)$. Se puede demostrar que los parámetros de la recta que mejor se ajusta a los datos están datos por:

$$m = \frac{x \cdot y - n\bar{x}\bar{y}}{x \cdot x - n\bar{x}^2}, \quad b = \bar{y} - m\bar{x},$$

donde $\cdot$ representa el producto punto, y $\bar{x}$ denota el promedio de *x*.

Escriba una función que tome *x*, *y, n* como argumentos y que calcule y devuelva *m* y *b*. Escriba un programa que genere un conjunto de 1,000,000 de datos de prueba con el modelo $y_i = 3x_i - 5 + e_i$, donde $x_i \sim U(-100,100)$ y $e_i \sim U(-10,10)$, y luego utilice la función anterior para recuperar *m* y *b* a partir de los datos de prueba.

# Tarea #10 – Parte 2

- Escriba una función llamada `genera(F, m, n, p)` que genere una matriz binaria aleatoria *F* de tamaño *m* x *n* donde cada elemento sea igual a 1 con probabilidad *p*.

- Escriba una función llamada `imprime(F, m, n)` que borre la pantalla (usando `system("cls")`) y luego imprima la matriz binaria *F* de tamaño *m* x *n*, donde los ceros se imprimirán como espacios en blanco y los unos como asteriscos.

- Escriba una función llamada `evolucion(F, G, m, n)` que tome como parámetros dos matrices de enteros, *F* y *G*, de tamaño *m* x *n,* donde cada elemento es binario (0 ó 1). Digamos que si un elemento vale 1, la celda correspondiente está "viva". La función debe calcular $G_{i,j}$ a partir de los "vecinos" de $F_{i,j}$, considerando las siguientes reglas:

    1. Si una celda viva tiene menos de dos vecinos vivos, entonces morirá.
    2. Si una celda viva tiene mas de tres vecinos vivos, entonces morirá.
    3. Una celda viva continuará en ese estado solamente si tiene dos o tres vecinos vivos.
    4. Una celda muerta cobrará vida si tiene tres vecinos vivos, de lo contrario continuará en ese estado.

- Usando las tres funciones anteriores, escriba un programa que implemente el juego de la vida: el programa iniciará mostrando una matriz binaria aleatoria, y luego, de manera iterativa, encontrará una nueva matriz realizando una evolución, la mostrará en pantalla, y esperará a que el usuario presione la tecla Enter (usando la función `getchar()`). La iteración continuará hasta que el usuario presione la tecla 'x' (seguida de Enter).

SESION 11

# MANEJO DE ARCHIVOS

# Flujos de entrada y salida en C++

- En C++, las operaciones de entrada y salida se manejan a través objetos llamados *flujos.*

- Existen tres librerías estándar para el manejo de flujos:
  - <iostream> para entrada desde y salida hacia consola
  - <fstream> para entrada desde y salida hacia archivos
  - <sstream> para entrada desde y salida hacia objetos tipo `string` (cadenas de caracteres).

# Flujos de archivos

- La librería <fstream> define los elementos necesarias para el manejo de archivos. Los mas importantes son:

  - `ofstream` : clase de datos utilizada para escritura de archivos.

  - `ifstream` : clase de datos utilizada para lectura de archivos.

# Apertura y cierre de archivos

- Para acceder a un archivo (escritura o lectura) es necesario siempre abrir primero el archivo mediante la función `open()`, que recibe como parámetro el nombre del archivo.

- De igual manera, una vez que finaliza el acceso al archivo, es necesario cerrarlo mediante la función `close()`.

- Ejemplo: creación de un archivo de salida

```
ofstream of;
of.open("archivo.txt");
of << "Hola mundo!" << endl;
of.close();
```

# Escritura de archivos de texto

- Como se observó en el ejemplo anterior, la escritura de datos en un archivo de texto se realiza mediante el operador de inserción `<<`, al igual que con la instrucción `cout`.

# Ejemplo

- Muchos programas como Excel o Matlab pueden leer archivos de datos en forma de matriz, los cuales son fáciles de exportar desde un programa en C++.

```
int main() {
   float x, y, r;
   int i, int n = 100;
   ofstream of("datos.txt");
   for (i = 0; i < n; i++) {
       x = 10.0 * rand() / RAND_MAX – 5.0;
       r = 1.0 * rand() / RAND_MAX - 0.5;
       y = 0.5 * x - 3.0 + r;
       of << x << "\t" << y << endl;
   }
   of.close();
}
```

# Ejemplo

- Muchos programas como Excel o Matlab pueden leer archivos de datos en forma de matriz, los cuales son fáciles de exportar desde un programa en C++.

```
int main() {
   float x, y, r;
   int i, int n = 100;
   ofstream of("datos.txt");
   for (i = 0; i < n; i++) {
        x = 10.0 * rand() / RAND_MAX - 5.0;
        r = 1.0 * rand() / RAND_MAX - 0.5;
        y = 0.5 * x - 3.0 + r;
        of << x << "\t" << y << endl;
   }
   of.close();
}
```

El archivo puede abrirse desde el momento de declararlo

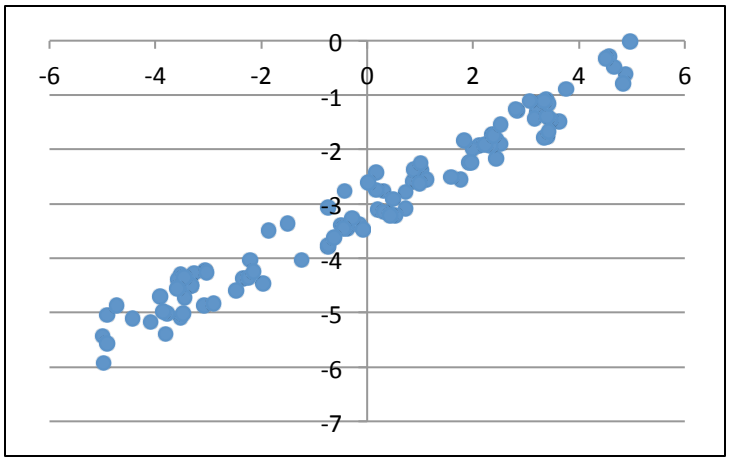
# Ejemplo

- Muchos programas como Excel o Matlab pueden leer archivos de datos en forma de matriz, los cuales son fáciles de exportar desde un programa en C++.

```
int main() {
   float x, y, r;
   int i, int n = 100;
   ofstream of("datos.txt");
   for (i = 0; i < n; i++) {
        x = 10.0 * rand() / RAND_MAX - 5.0;
        r = 1.0 * rand() / RAND_MAX - 0.5;
        y = 0.5 * x - 3.0 + r;
        of << x << "\t" << y << endl;
   }
   of.close();
}
```

El archivo puede abrirse desde el momento de declararlo

Los datos generados pueden graficarse en Excel fácilmente

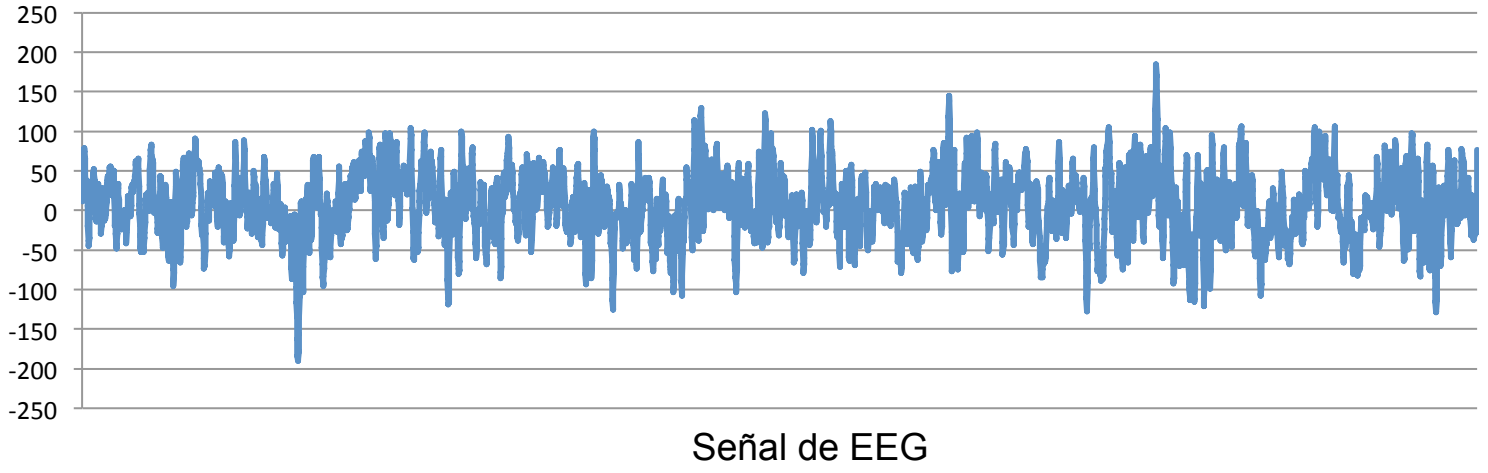


# Lectura de archivos de texto

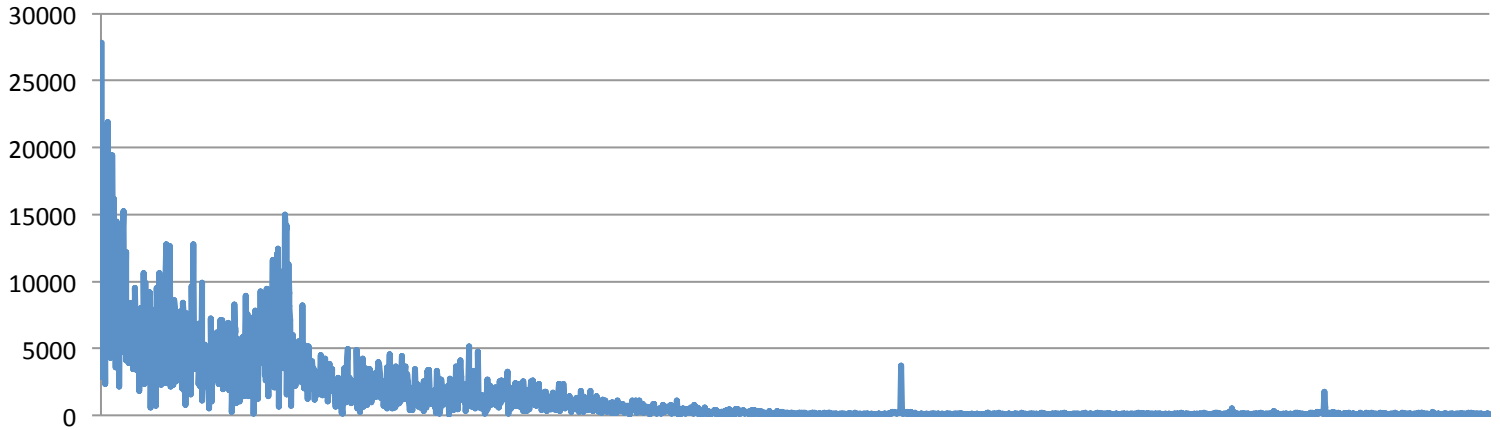
- De manera similar, podemos leer de un archivo de texto al igual que se hace con la instrucción `cin`:

```
int main() {
   int n, k, N = 4096;
   float x[N], X[N * 2];
   ifstream in("eeg.txt");
   ofstream out("espectro.txt");
   for (n = 0; n < N; n++) { in >> x[n]; }
   fft(x, X, N);
   for (k = 0; k < N/2; k++) {
       out << sqrt(X[k] * X[k] + X[k+N] * X[k+N]) << endl;
   }
   return 0;
}
```

# Lectura de archivos de texto




Señal de EEG




Espectro de frecuencias de la señal

# Algunas funciones adicionales

- El operador `!` puede utilizarse para saber si un archivo se abrió correctamente. En caso de haber algún problema, este operador devuelve verdadero.

- Es posible leer una línea completa de un archivo de texto y almacenarla en una cadena, mediante la función `getline(char *s, int n)`.

- Para detectar si se ha llegado al final del archivo, se puede utilizar la función `eof()`.

# Ejemplo

```
// Esta función imprime el contenido de un archivo
void muestra_archivo(char *s) {
   int n = 65536;
   char temp[n];
   ifstream in(s);
   if (!in) return;
   while (!in.eof()) {
        in.getline(temp, n);
        if (!in.eof()) { cout << temp << endl; }
   }
   in.close();
}

int main() {
   muestra_archivo(“ejemplo.cpp”);
   return 0;
}
```

# Ejemplo

```
// Esta función imprime el contenido de un archivo
void muestra_archivo(char *s) {
   int n = 65536;
   char temp[n];
   ifstream in(s);
   if (!in) return;
   while (!in.eof()) {
        in.getline(temp, n);
        if (!in.eof()) { cout << temp << endl; }
   }
   in.close();
}

int main() {
   muestra_archivo("ejemplo.cpp");
   return 0;
}
```

# Ejemplo

```
// Esta función imprime el contenido de un archivo
void muestra_archivo(char *s) {
   int n = 65536;
   char temp[n];
   ifstream in(s);
   if (!in) return;
   while (!in.eof()) {
        in.getline(temp, n);
        if (!in.eof()) { cout << temp << endl; }
   }
   in.close();
}

int main() {
   muestra_archivo("ejemplo.cpp");
   return 0;
}
```

# Ejemplo

```
// Esta función imprime el contenido de un archivo
void muestra_archivo(char *s) {
   int n = 65536;
   char temp[n];
   ifstream in(s);
   if (!in) return;
   while (!in.eof()) {
        in.getline(temp, n);
        if (!in.eof()) { cout << temp << endl; }
   }
   in.close();
}

int main() {
   muestra_archivo("ejemplo.cpp");
   return 0;
}
```

Leer la siguiente línea del archivo y almacenarla en la cadena `temp`

# Práctica #1

- Haga un programa que lea una serie de números enteros desde un archivo de texto. El primer número en el archivo indica cuántos datos más hay en el archivo. El programa debe almacenar todos los datos (excepto el primero) en un arreglo *X* de tamaño adecuado.

- Luego, el programa debe generar otro arreglo *Y* que contenga los mismos datos que *X*, pero sin repetirse; es decir, si un número aparece múltiples veces en *X*, aparecerá solo una vez en *Y*.

- Guarde el contenido de *Y* en un archivo de texto con el mismo formato que el archivo de entrada; es decir, escribiendo primero el número de datos, y posteriormente los datos.

# Práctica #1

- Ejemplo de ejecución:

Archivo de entrada

```
12
5
8
7
8
9
2
3
4
5
4
5
8
```

Archivo de salida

```
7
5
8
7
9
2
3
4
```

# Tarea #11 – Parte 1

- Escriba una función

  `escmat(float *x, int m, int n, char *s)`

  que escriba una matriz `x` de tamaño `m` x `n` en el archivo con nombre `s`. Cada renglón de la matriz debe escribirse en un solo renglón del archivo, y los valores de cada renglón deben separarse por tabuladores `'\t'`.

- Escriba una función

  `leemat(float &*x, int &m, int &n, char *s)`

  que lea una matriz `x` desde un archivo `s` en el formato utilizado por `escmat()`. Note que el tamaño de la matriz es desconocido y debe obtenerse a partir del mismo archivo (por ejemplo, contando el número total de elementos y el número de renglones). Por lo mismo, el espacio para la matriz debe reservarse (pero no liberarse) de manera dinámica dentro de la función.

- Para probar ambas funciones, genere una matriz de números aleatorios. Guárdela en un archivo y luego recupere la matriz a partir del archivo y verifique que es idéntica a la original. Así mismo, intente leer y exportar matrices en este formato desde Excel.

# Tarea #11 – Parte 2

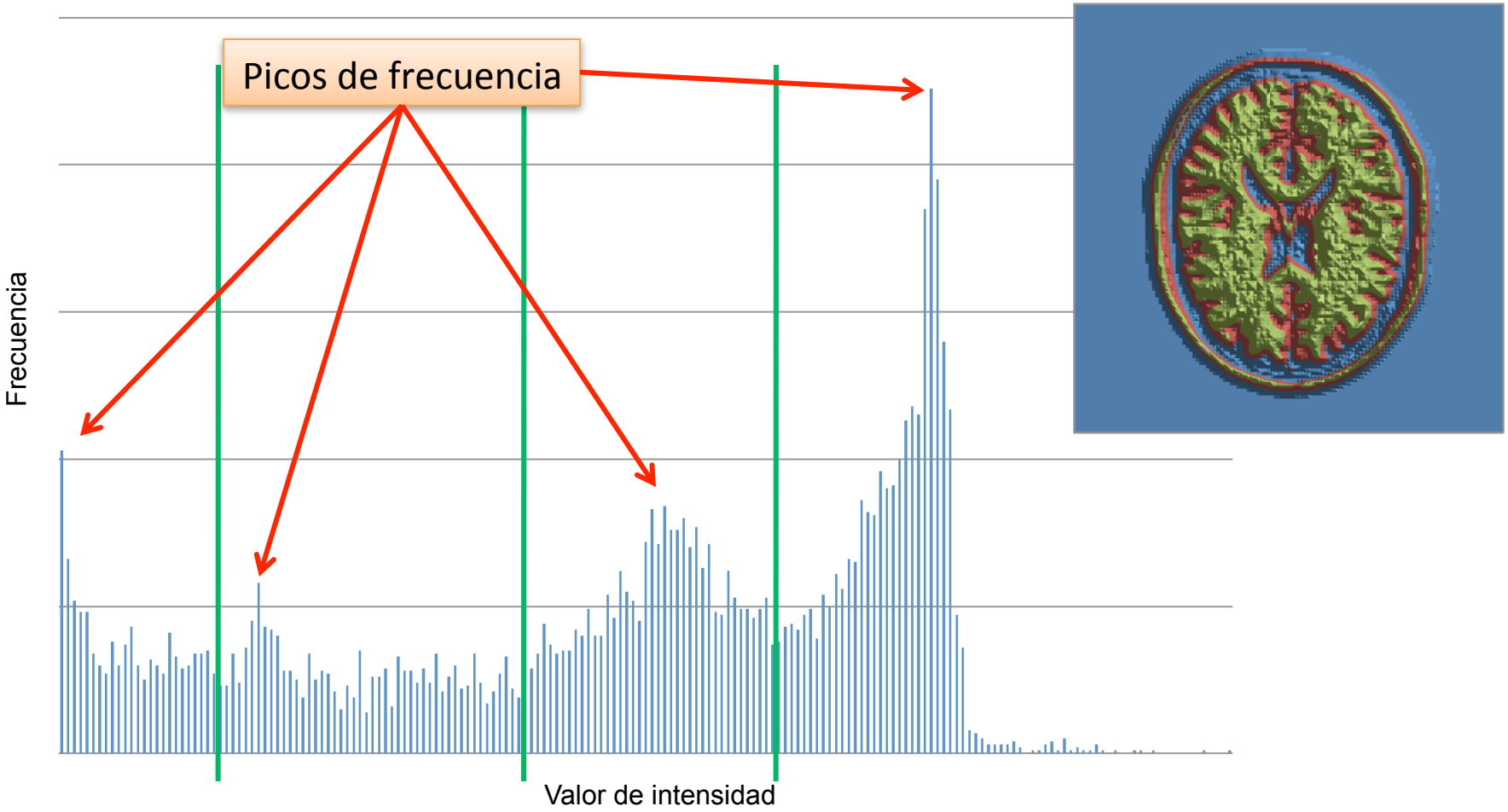
- Una imagen digital puede verse como una matriz donde cada elemento representa el nivel de intensidad o brillo del pixel correspondiente.

- El archivo mri.txt contiene una imagen de resonancia magnética cerebral en el formato utilizado por la función leemat(). Importe este archivo en Excel y grafique los datos utilizando el tipo de gráfico Superficie-Contorno. Observe cómo los distintos tejidos se muestran con un color distinto.

- Elabore un programa en C++ en el que se cargue la imagen mri.txt (como una matriz) y calcule el histograma h de los niveles de intensidad (los cuales son enteros entre 0 y 255). Guarde el histograma en otro archivo y grafíquelo en Excel eliminando el primer dato.

- Observe que existen cuatro "picos" en el histograma. Estime a simple vista los valores de intensidad que separan estos picos. Suponga que uno de los picos está entre los valores $a$ y $b$. Elabore un programa que modifique la imagen mri.txt haciendo cero cualquier valor de intensidad que quede fuera del rango $[a,b)$. Guarde la nueva imagen en otro archivo (e.g., mri2.txt) y muéstrela en Excel. Haga lo mismo para los cuatro picos.

# Tarea #11 – Parte 2




Histograma de mri.txt