Práctico 2 - Tipos, Operadores y Expresiones

Ejercicio 1 - Tamaño a medida

Escriba un programa que tome un número entero como argumento desde la línea de comandos, e indique al usuario cuál es el tipo de entero más pequeño capaz de representarlo correctamente. Por ejemplo, el número 1022 no entra en un `char`, pero sí en un `int`. Se sugiere utilizar la función `atol` de `<stdlib.h>` para convertir el argumento de línea de comandos a un `long int` y luego verificar si cabe en tipos más pequeños, asignándolo a esos tipos y verificando que se preserva el valor. Una alternativa es medir la cantidad de bits necesaria para representar al número en cuestión (qué función matemática puede servir para ello?) y luego utilizar el operador `sizeof` para seleccionar el tipo que mejor calce.

Ejercicio 2 - Inicialización de variables automáticas

Compile y ejecute el siguiente código. Explique las diferencias observadas.

```
#include <stdio.h>

int k; /* variable global */

int f() {
  int a;
  a++;
  return a;
}

int g() {
  int a = 0;
  a++;
  return a;
}

main() {
        int i;
        printf("k=%d\n",k);
        for (i = 0; i < 10; i++) {
                printf("a en f() vale %d\n",f());
        }
        for (i = 0; i < 10; i++) {
                printf("a en g() vale %d\n",g());
        }
}
```

Ejercicio 3 - Operador módulo

El operador módulo entero `%` implementa en hardware la funcionalidad vista en la función `MOD` descrita en el práctico 1. Para probarlo, supongamos que quiere imprimir una guía para saber en qué

columna está una letra en la pantalla. Escriba un programa que pueda imprimir cadenas de caracteres de largo n arbitrario, formada por repeticiones concatenadas del patrón `‘‘+----’’`. Por ejemplo, para $n = 12$ la cadena sería `‘‘+----+----+--’’`. **El programa debe consistir en un sólo bucle; no vale dos!**

Ejercicio 4 - Buffer circular (†)

Los buffers circulares son estructuras de datos muy sencillas pero muy importantes en la programación de algoritmos relacionados con la ingeniería eléctrica, en particular de filtros digitales. Un buffer es, en general un espacio de almacenamiento temporal de datos de tipo “FIFO” (First In First Out), es decir, los datos se sacan del buffer en el orden en que fueron introducidos. Generalmente, un buffer consiste en un arreglo de un cierto largo máximo (llamémosle `NMAX`), un índice al último lugar que se escribió (llamémosle `iin`) y un índice al último lugar que se leyó (`iout`). Las operaciones que pueden realizarse sobre un buffer son: guardar un nuevo dato, y tomar el dato más viejo. Según la definición de los arreglos de C, el índice lineal más bajo en tal buffer es `0` y el más alto es `NMAX-1`.

Los buffers circulares tienen la particularidad de que los índices se consideran *módulo NMAX*, es decir, si uno intenta acceder a un buffer en la posición `NMAX+1`, el lugar en memoria que será leido es el del arreglo en la posición `1`. De la misma manera, si se intenta acceder a un índice negativo, por ejemplo `-1`, el lugar del arreglo que será devuelto es `NMAX-1`.

a) Considere un arreglo `buffer` de caracteres de largo `NMAX=4`, y un entero `i`

b) Escriba una o más expresiones en C para almacenar un nuevo valor `char a` en la posición `i+1` del buffer, incrementando en el proceso el valor de `i`.

c) Escriba una o más expresiones en C para *leer* el valor del buffer en la posición `i-k`, (sin actualizar `i`), donde `k` es un entero arbitrario.

d) Intente repetir las dos partes anteriores utilizando otras expresiones de C.

e) Al respecto de las dos partes anteriores, existe alguna ventaja en que `NMAX` sea una potencia de `2`? Qué podría hacerse en ese aso?

f) Escriba un programa que defina las variables anteriormente mencionadas, rellene el buffer `buffer` con el caracter `'X'`, inicialice `i=0`, y luego uno a uno, alimente los caracteres del primer argumento de la linea de comandos en el buffer y luego imprima el caracter del buffer en la posición `i-3`. (Debería obtenerse una salida “retardada” en donde los primeros tres caracteres impresos sean 'X'.)

Ejercicio 5 - Operadores lógicos

Escriba una expresión lógica que implemente la función lógica $z = f(w, x, y)$ dada por la siguiente tabla:

0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

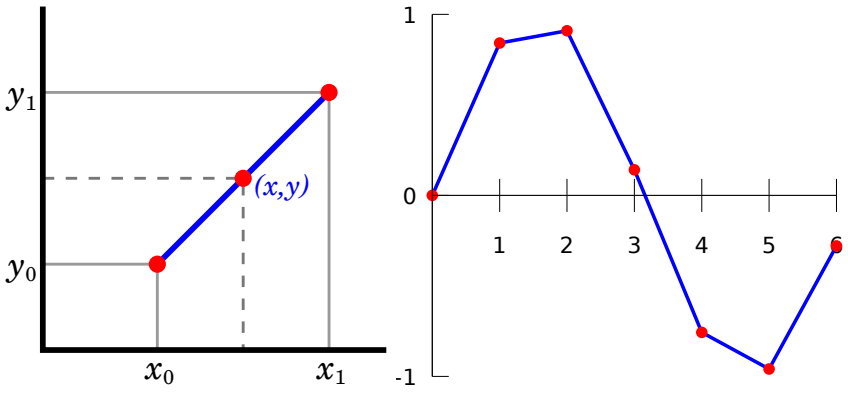

Figura 1: Interpolación lineal.

Ejercicio 6 - Operadores de bit

Los operadores de bit `|`,`&`,`^`,`~`,$\gg$,$\ll$ son muy imporantes en aplicaciones de bajo nivel. El siguiente ejercicio pone a prueba su uso con pequeños problemas.

a) Representación binaria Escriba un programa que tome un entero sin signo e imprima su representación en base binaria.

b) Enmascarado Escriba un programa que tome tres argumentos enteros. El primero es un entero cualquiera n, y los siguientes dos, llamémosles a y b, sirven para delimitar una máscara binaria que se usará para apagar todos los dígitos binarios del número n que estén por debajo de la posición a y a partir de la posición b inclusive, y luego imprima el resultado. Por ejemplo, si tenemos el número $n = 45$ (en binario 00101101), $a = 2$ y $b = 5$, el resultado sería 00001100.

c) Encriptado sencillo Escriba un programa que tome dos cadenas de texto: una larga, especificando un `texto` a encriptar, y una corta, a usar como `clave` de encriptación. La encriptación se realiza haciendo `xor` de **los primeros 6 bits** de cada letra en el `texto` a encriptar con una letra correspondiente de la `clave`. Al principio se alinean las dos cadenas de texto y cuando el índice en el texto original supera al de la clave, se retoma desde su comienzo. A modo de ejemplo, si el texto es "hasta la vista baby" y la clave es "terminator", la correspondencia es la siguiente:

```
HASTA LA VISTA, BABY!
TERMINATORTERMINATORT
```

y el resultado encriptado:

```
\DAYH.MU/D]VFL%.CUMK5
```

Si todo sale bien, el texto encriptado resultante (que puede contener caracteres no imprimibles) puede ser recuperado aplicando la misma función sobre él, usando la misma clave.

Nota: Se sugiere primero resolver el problema de manejo de bits y luego atacar el resto. A modo de ayuda, recuerde que puede especificar constantes numéricas en base hexadecimal poniendo como sufijo `0x`. Por ejemplo, `0x7f` es el decimal `127`, binario `01111111`.

Ejercicio 7 - Interpolación de funciones y LUTs (†)

Una de las relaciones de compromiso más comunes en la ingeniería, cuando se trata de implementar software, es *memoria vs. velocidad*. Uno de los ejemplos más clásicos de esto son las *tablas de funciones* o *LUT* (del inglés Look-Up Table). La idea es tener almacenados en memoria los valores de una función

$f(x)$ costosa de calcular (por ejemplo, $\cos(x)$), de modo que aproximar su valor mediante esa tabla resulte mucho menos costoso que calcularlo de cero. En general, la tabla almacena $f(x)$ para un conjunto de N valores consecutivos de x dentro de cierto rango, por ejemplo $\mathcal{X} = \{0, \delta, 2\delta, \ldots, (N-1)\delta\}$. Valores de $f(x)$ fuera de ese conjunto deberán ser *interpolados* en base a valores de $f(x)$ que sí estén almacenados.

En este punto, surgen varias alternativas: Por un lado, para un mismo tamaño de tabla, se obtendrán mejores aproximaciones si se utilizan técnicas de interpolación más sofisticadas. La técnica más burda es el valor más cercano (devolver $f(\hat{x})$ tal que $|x - \hat{x}|$ es mínimo). En la otra punta tenemos interpolación cúbica, polinómica, o Splines. Por otro lado, si se fija un método de interpolación, la aproximación mejorará a medida que la tabla sea más "fina" (es decir, δ sea menor, y la tabla tenga más puntos). Para un mismo rango a representar, esto claramente requiere más memoria.

La idea de este ejercicio es la de implementar distintas variantes de LUTs, con distintos métodos de interpolación, y ver su resultado en términos de error cuadrático.

a) Implemente un programa que reciba como argumento un entero N y luego:

1. declara un arreglo de valores de punto flotante de nombre `lut` y tamaño `N`
2. Rellene dicho arreglo con un período de la función `cos(x)`, de modo que el primer elemento de `lut` contenga el valor de $\cos(0)$ y el último $\cos(2\pi(N-1)/N)$. (se sugiere aproximar π como acos(−1.).) Cuál es el valor de δ en este caso?
3. Estime el siguiente error de aproximación (aproximando la integral como una suma de Riemann en pasos $\Delta x = 2\pi/10000$),

$$e = \int_{x=0}^{x=2\pi} (cos(x) - c\bar{o}s(x))^2 dx,$$

   donde $c\bar{o}s(x)$ es la aproximación dada por la interpolación de vecinos más cercanos en la tabla,

$$c\bar{o}s(x) = \text{lut}[\hat{i}],\ \hat{i} = \arg\min_i \{|x - i\delta| : i = 0, 1, \ldots, N-1\},$$

   es decir, el valor más cercano a x en el conjunto $\mathcal{X}$.
4. Imprima el valor de N, el tipo de interpolación usada (en este caso vecino más próximo) y el valor de e.

Ejecute el programa anterior para varios valores de N, observando su salida.

b) Repita el ejercicio anterior utilizando interpolación lineal. En este caso

$$c\bar{o}s(x) = [\hat{x}],\ \hat{x} = (x_1 - x)\text{lut}[x_0] + \text{lut}[x_1](x - x_0),$$

donde

$$x_0 = \text{lut}[\hat{i_0}],\ \hat{i}_0 = \arg\max_i \{i\delta \leq x, i = 0, 1, \ldots, N-1\}$$
$$x_1 = \text{lut}[\hat{i_1}],\ \hat{i}_1 = \arg\min_i \{i\delta > x \in, i = 0, 1, \ldots, N-1\},$$

es decir, los dos enteros mas cercanos a x (uno por arriba, el otro por abajo) en la tabla.

Ejercicio 8 - Frecuencias, histogramas y probabilidades empíricas (*)

Posiblemente la forma más sencilla de estimar probabilidades es a través de frecuencias empíricas, es decir, el conteo de ocurrencias de los distintos eventos aleatorios posibles a lo largo de un número grande de repeticiones de un cierto experimento.

Supongamos que en experimento hay m posibles eventos, y que la ocurrencia de cada evento es representada como un entero W entre 0 y $m-1$. Nuestros datos de entrada consisten en un vector de largo n, $\mathbf{x}$, donde $x_i = j$ indica que el experimento i-ésimo dió como resultado $W = j$.

Definamos como $f_j, 0 \leq j < m$ a la cantidad de veces que el evento $W = j$ ocurrió en $\mathbf{x}$. Al vector de valores $\mathbf{f} = (f_0, f_b, \ldots, f_{m-1})$ lo denominamos *histograma*. Dado el histograma histograma $\mathbf{f}$, la probabilidad del evento $W = j$ puede ser estimada como $P(W = j) = f_j/n$.

a) Escriba una función `int calcular_histograma(int x[], int m, int n, int f[])` que tome una serie de n datos en `x` con valores entre 0 y $m-1$, y calcule las frecuencias de cada uno de los posibles m eventos en el arreglo de largo m, `f`.

b) Complete el siguiente programa (el archivo **C** correspondiente se llama `histograma.c` y está disponible en la página del curso) con la función que usted acaba de escribir, ejecútelo y lea con atención la salida del programa.

```
#include <stdio.h>

#define N 10
#define M 3
int numeritos[N] = {0,1,2,1,2,0,0,1,2,2};

void calcular_histograma(int datos[], int m, int n, int f[]) {
  /* RELLENAR AQUI */
}

int main() {
  int i;
  int f[M];   /* frecuencias */
  float P[M]; /* probabilidades */

  calcular_histograma(numeritos,M,N,f);

  for (i = 0; i < M; ++i) {
    P[i] = f[i]/N;
    printf("frecuencia f[%d]=%d -> probabilidad P[%d]=%f\n",i,f[i],i,P[i]);
  }
}
```

c) Verifique que las frecuencias fueron bien calculadas (debería ser $f_0 = 3$, $f_1 = 3$, $f_2 = 4$). ¿Qué sucede con las probabilidades estimadas? ¿Qué está mal? Modifique el programa anterior para que las probabilidades se calculen correctamente.

d) Modifique su función `calcular_histograma` de modo que si llegara a ocurrir que $x_i > m-1$ (para el cual no está previsto registrar la ocurrencia), se incremente en su lugar la frecuencia del evento $W = m-1$. De la misma manera, si llegara a ocurrir que $x_i < 0$, es la frecuencia del evento $W = 0$ la que debe incrementarse. Esta estrategia es conocida como "clipping" o "recorte", y puede resumirse

de la siguiente manera:

$$y = \text{clip}_{a,b}(x) = \begin{cases} a & , \quad x < a \\ x & , \quad a \leq x \leq b \\ b & , \quad b < x \end{cases}$$

e) Desafío intente escribir la función $\text{clip}_{a,b}(x)$ en una sóla línea, usando el operador ternario de asignación condicional `?:`. Este operador es en general mucho más eficiente que utilizar sentencias `if-else`.

Ejercicio 9 - Representación de matrices

Hay esencialmente dos maneras de representar una matriz bidimensional $\mathbf{X}$ de tamaño $m \times n$ en C. La más intuitiva es definirla como un arreglo bidimensional

```
float X[m][n];
```

Una manera alternativa, menos intuitiva pero muchas veces más conveniente, es la de definir dicha matriz en un arreglo unidimiensional de largo mn,

```
float X[mn];
```

y definir una correspondencia entre los índices bidimensionales de la matriz $(i, j), 0 \leq i < m, 0 \leq j < n$ y los índices unidimiensionales $0 \leq k < mn$. Una manera posible de hacer esto último es "por filas", es decir, los primeros n elementos de `X` (`X[0]` a `X[n-1]` inclusive) corresponden a la primera fila de la matriz $\mathbf{X}$, los siguientes n elementos ((`X[n]` a `X[2n-1]` inclusive) a la segunda fila , etc.

Esta es la forma en que por ejemplo se suelen almacenar imágenes digitales en memoria para su posterior procesamiento.

a) Implemente una función `double elemento(double X[],int i, int j)` que devuelva el elemento (i, j) de una matriz bidimensional almacenada por filas en un arreglo unidimiensional `X`.

b) (*) Modifique la función anterior de modo que si se especifican coordenadas fuera de rango, éstas sean recortadas mediante la función clip descrita en el ejercicio anterior, es decir que el valor devuelto por la función sea $\mathbf{X}(i', j')$ donde $i' = \text{clip}_{0,m}(i)$ y $j' = \text{clip}_{0,n}(j)$. (Esto se utiliza mucho en procesamiento de imágenes para "estirar los bordes" de la imagen).