Introducción a C*

Objetivos

- Introducción a C*.
- Mostrar cómo se definen las funciones auxiliares y el programa principal.
- Uso de bibliotecas estándar de entrada/salida (leer desde el teclado/escribir en pantalla).

Ejercicio 1 Promedio clase

Suponga que a los estudiantes de programación 2 se les dice que su calificación final será el promedio de las cuatro calificaciones más altas de entre las cinco que hayan obtenido en el curso.

(a) Escribir una función llamada **PromClase** con cinco parámetros de entrada (las calificaciones obtenidas) que realice dicho cálculo y lo devuelva.

(b) Escribir un programa principal (`main ()`) que permita ejecutar la función **PromClase**. Dicho programa deberá leer de la entrada estándar (teclado) 5 calificaciones, invocar al procedimiento **PromClase** con dichos parámetros, y finalmente mostrar en la salida estándar (pantalla) el resultado.

Ejercicio 2 Primos

Escriba un procedimiento que calcule e imprima en pantalla todos los números primos entre dos enteros positivos A y B cualesquiera.

Ejercicio 3 Ocurrencias

Se quiere implementar una función que cuente la cantidad de veces que una letra aparece en una frase. La frase se representa como un **arreglo de caracteres**, y dado que se conoce que el largo máximo de una frase es de 100 caracteres, la frase se implementa como char frase[100]. Usando esta representación escriba una función **Ocurrencias** que recibe una **frase**, un natural llamado **largo** que representa el número de caracteres en la frase, y el caracter a buscar (almacenado en la variable **letra**), y devuelve el número de ocurrencias del carácter **letra** en el arreglo **frase**.

Ejercicio 4 Es Palíndrome

Considere ahora que la **frase** se representa como un **arreglo de caracteres** implementado como **char *frase**.

(a) ¿Qué diferencias hay entre esta representación y la que utilizó en el Ejercicio 3?

(b) Escriba una función *EsPalindrome* que recibe una frase representada como un puntero a `char` y devuelve TRUE si la misma es un palíndrome, FALSE en otro caso.

Ejercicio 5 Insertar Punto

Un punto en un plano cartesiano se representa como un par ordenado de números, de forma que el primer número corresponde a la coordenada horizontal x y el segundo número a la coordenada vertical y.

Los siguientes tipos definen los conceptos *punto* y *colección de puntos*. La colección se representa con un arreglo con tope.

```
typedef struct rep_punto {
  int coordX, coordY;
} Punto;

typedef struct rep_colPuntos {
  int capacidad;
  int cantidad;
  Punto * arregloPuntos;
} ColPuntos;
```

(a) Implemente una función que inserte un punto al final de una colección de puntos:

```
void insertarPunto(ColPuntos & colPtos, Punto pto);
```

Si la cantidad de puntos es igual a la capacidad del arreglo la operación no tiene efecto.

(b) Defina e implemente una función que dada una colección de puntos devuelva el valor de la menor coordenada `x` de sus puntos.

(c) Defina e implemente una función que dados dos puntos devuelva el punto medio del segmento determinado por ambos.

Ejercicio 6 Ordenar arreglo

Escriba una función que recibe un arreglo de enteros y devuelve un nuevo arreglo que contiene a los elementos del primero en orden ascendente. Indique qué algoritmos de ordenación utiliza.

Se presenta el siguiente fragmento de pseudocódigo del **algoritmo de Ordenamiento por selección (Selection Sort)**, como ayuda para la realización del ejercicio:

```
// Declaración e inicialización del arreglo
Declarar un arreglo de n elementos.
Para cada entrada i del arreglo:
    Leer un entero de la entrada estandar.
    Almacenarlo en la entrada i del arreglo.

// Algoritmo de ordenamiento
Para cada entrada i del arreglo:
    Buscar el mínimo elemento e_min de las entradas i en adelante.
    Intercambiar dicho elemento e_min con el de la posición i.

// Impresión de resultado
Para cada entrada i del arreglo:
    Imprimir el elemento en la entrada i.
```