



Maestría en Ciencias y Tecnologías de la Información

Inteligencia Artificial Aplicada

# PRACTICA 1: Problemas básicos para introducir el mundo de C

Alumno Gustavo Alfredo Flores Pérez
Dr. Benjamín Moreno Montiel

México D.F.
Agosto 2022

# Contenido

# INTRODUCCIÓN

Aunque Python permanece en la cima, es seguido de cerca por C. Esto sin considerar los lenguajes combinados que utilizan los sistemas embebidos, por lo que lo podemos considerar como el lenguaje más utilizado[1], además de que se pueden encontrar nativas de el en otros lenguajes.

Los apuntadores son variables que contienen las direcciones de memoria, a diferencia de una variable que se refiere directamente a un valor. Generalmente las usamos para almacenar direcciones que almacenan objetos, los apuntadores son mas eficientes en ese sentido.

Los apuntadores tienen un espacio de memoria por default de 8 bytes, ya que con eso pueden direccionar cualquier tipo de dato, aunque no se utilice completamente.

Operador de referencia El operador prefijo & obtiene la dirección de inicio de una variable. A esto también se le llama una referencia a la variable. Si declaramos int n entonces &n es una referencia a n.

Operador de des referencia El operador prefijo* obtiene la variable a la que se hace referencia. A esto también se le llama una desreferencia. En otras palabras, *&n es n.[2]

Para efectos de esta práctica la utilización de apuntadores nos permitió:

- Definir arrays de tamaño variable.
- Que las funciones devuelvan más de un valor.
- Definir estructuras de datos complejas (p.e. listas, árboles)

Así como en el caso de los arreglos de una dimensión, es posible representar los arreglos multidimensionales con una notación de punteros equivalente.

La programación modular nos permite seguir una estrategia de dividir el problema en varias secciones para poder llegar a la solución global del mismo, en c podemos aplicar esta estrategia mediante estructuras y prototipos de funciones.

Una estructura es una colección de uno a más tipos de elementos denominados campos, cada uno de los cuales puede ser un tipo de dato diferente como en la vida real una torta cubana.

DESARROLLO

Para el reconocedor de tokens y lexemas, metemos carácter por carácter los elementos dentro de un vector como se muestra en la siguiente figura.

```
if (simbolo != ' ' && simbolo != '\n')
{
    lexema[c] = simbolo; // Ingresa los caracteres al string
    c++;                 // Si encuentra un caracter se mueve a la sig. posición del vector
}
else
{
    printf("TOKEN = ");
    if (!(lexema[0] >= 48 && lexema[0] <= 57))
    {
        if (strcmp(lexema, "false") == 0 || strcmp(lexema, "true") == 0)
        { // Compara la cadena entre falso y verdadero
            printf("Booleano,           ");
        }
        else
        {
            printf("Cadena de caracter, ");
        }
    }
    else
    {
        char elem = '.';
        if (EstaEn(lexema, elem))
        { // Si hay algun valor difente de cero entonces es un flotante
            printf("Flotante,           ");
        }
        else
        {
            printf("Entero,             ");
        }
    }
    c = 0; // Restablecer el iterador de la cadena para re-construir otra cadena
    printf("LEXEMA := %s\n", lexema);
    cont++;
    free(lexema);
    lexema = (char *)malloc(sizeof(char) * n);
}
```

Dentro de ese vector se va distinguiendo entre letras y números de acuerdo con el código ascii y posteriormente en el caso de letras se identifica true o false para los booleanos, de forma análoga la estrategia en el caso de separar números se hace recorriendo el vector hasta encontrar un punto y de esta forma podemos distinguir entre un número y un flotante.

```c
int EstaEn(char *cadena, char c)
{
    /*Devuelve 1 si c esta en el string cad */
    // printf("Cadena = %s\n", token);
    while (*cadena != '\0')
    {
        if (*cadena == c)
        {
            return 1;
        }
        cadena++;
    }
    return 0;
}
```

```
Fichero: tokens.txt -> Existe -> (ABIERTO)
Los tokens y lexemas encontrados en el fichero tokens.txt fueron:

TOKEN = Cadena de caracter, LEXEMA := hola
TOKEN = Booleano,           LEXEMA := true
TOKEN = Booleano,           LEXEMA := false
TOKEN = Entero,             LEXEMA := 32
TOKEN = Flotante,           LEXEMA := 2.3
TOKEN = Cadena de caracter, LEXEMA := c
TOKEN = Cadena de caracter, LEXEMA := ture
TOKEN = Cadena de caracter, LEXEMA := adios
TOKEN = Entero,             LEXEMA := 22
TOKEN = Entero,             LEXEMA := 4
TOKEN = Flotante,           LEXEMA := 5.6
TOKEN = Cadena de caracter, LEXEMA := d
TOKEN = Cadena de caracter, LEXEMA := f
TOKEN = Cadena de caracter, LEXEMA := r
TOKEN = Cadena de caracter, LEXEMA := t
TOKEN = Booleano,           LEXEMA := false
TOKEN = Cadena de caracter, LEXEMA := abc
TOKEN = Entero,             LEXEMA := 6
TOKEN = Flotante,           LEXEMA := 6.7
TOKEN = Entero,             LEXEMA := 5
TOKEN = Entero,             LEXEMA := 5
TOKEN = Entero,             LEXEMA := 6
TOKEN = Entero,             LEXEMA := 8
TOKEN = Cadena de caracter, LEXEMA := g
TOKEN = Cadena de caracter, LEXEMA := rr

Se encontraron 25 elementos
```

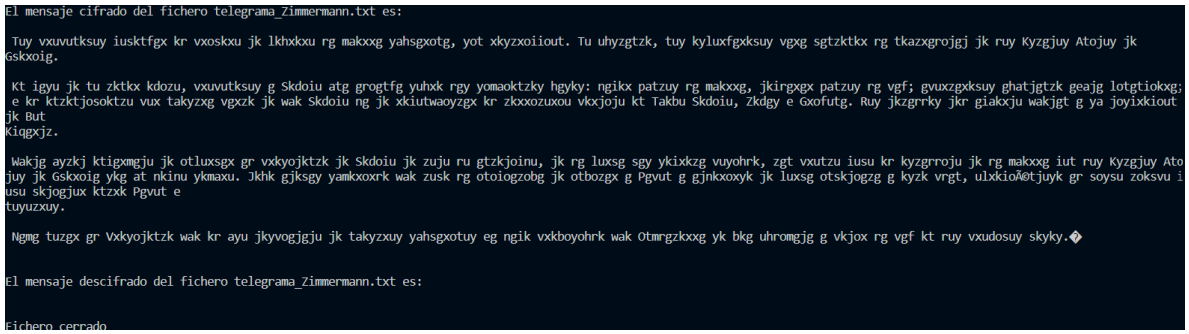
Cifrado cesar

En este caso la estrategia es similar en el caso anterior, se fue capturando carácter por carácter y realizando el cifrado al escribirlo en el vector de nombre cesar, esto se aplicó a Mayúsculas y minúsculas, en el caso de otro tipo de carácter se deja pasar sin ninguna modificación.

```
while (!feof(fichero))
{
    ascii = fgetc(fichero);
    if (ascii != ' ' && ascii != '\n')
    {
        if (ascii >= 65 && ascii <= 90)
        {
            cesar[i] = ascii;
            cesar[i] = (((cesar[i] - 65) + key) % 26) + 65;
            i++;
        }
        else if (ascii >= 97 && ascii <= 122)
        {
            cesar[i] = ascii;
            cesar[i] = (((cesar[i] - 97) + key) % 26) + 97;
            i++;
        }else{
            cesar[i] = ascii;
            i++;
        }
    }
    else
    {
        if (ascii == '\n')
        {
            printf("\n%s", cesar);
        }
        else
        {
            printf(" %s", cesar);
        }

        i = 0;
        free(cesar);
        cesar = (char *)malloc(sizeof(char) * 20);
    }
}
```

Los resultados se muestran a continuación:



Árbol Ordenado

- Recorrido del árbol.

```c
int main(int args, char *argv[])
{
    int niveles, nivel;
    int i, j, k, valores;
    srand(time(NULL));
    nivel = 0;
    printf("\nProporciona el factor de ramificacion del arbol:\n");
    scanf("%d", &fact_ram);

    printf("\nCuantos niveles del arbol quieres generar??\n");
    scanf("%d", &niveles);

    Nodo_BFS *nodo_inicial;
    Nodo_BFS *nodo;
    Nodo_BFS **Hijos = (Nodo_BFS **)malloc(fact_ram * sizeof(Nodo_BFS *));
    for (i = 0; i < fact_ram; i++)
    {
        Hijos[i] = (Nodo_BFS *)malloc(fact_ram * sizeof(Nodo_BFS));
    }

    valores = rand() % 101;
    nodo_inicial = creanodo_arbol(valores);
    arbol = nodo_inicial;

    Encolar(&Cola, nodo_inicial, 1);
    Encolar(&E_A, nodo_inicial, 0);
    nivel++;
    int exp = 0;
    while (TAM_COLA != 0)
    {
        for (k = 0; k < pow(fact_ram, exp); k++)
        {
            nodo = Desencolar(&Cola);
            if (nivel <= niveles)
            {
                for (i = 0; i < fact_ram; i++)
```

```
            {
                valores = rand() % 101;
                Hijos[i] = creanodo_arbol(valores);
                Encolar(&Cola, Hijos[i], 1);
                Encolar(&E_A, Hijos[i], 0);
            }
            nodo = set_hijos(Hijos, nodo);
        }
    }
    exp++;
    nivel++;
}
j = 0;

for (i = 0; i <= niveles; i++)
{
    if (i == 0)
    {
        printf("Los datos del arbol con sus respectivos datos son los siguientes\n");
        printf("Padre\n\tDato=%d\n\t", E_A->info->dato);
        printf("Direccion de Memoria del Padre= %p\n\t", E_A->info);
        printf("Direccion de Memoria del arreglo de Hijos=%p\n", E_A->info->hijos);
        E_A = E_A->sig;
        j++;
    }
    else
    {
        printf("\n\nLos hijos del nivel %d con su respectiva padre son:\n", i);
        printf("\tDireccion de Memoria del arreglo de los Hijos=%p\n", E_A->info->hijos);
        k = 0;
        while (k < pow(fact_ram, j))
        {
            printf("\n\tHijo[%d][%d]\n\tDato=%d\n", i, k + 1, E_A->info->dato);
            printf("\tDireccion de Memoria del Hijo[%d][%d]=%p\n", i, k + 1, E_A->info);
            printf("\tDireccion de Memoria de su Padre=%p\n", E_A->info->padre);
            E_A = E_A->sig;
            k++;
```

- Construcción del árbol

```
Nodo_BFS *creanodo_arbol(int elem)
{
    Nodo_BFS *aux;
    aux = (Nodo_BFS *)malloc(sizeof(Nodo_BFS));
    aux->dato = elem;
    aux->sig = NULL;
    return (aux);
}
```

```
Nodo_BFS *Desencolar(Nodo_Cola **Cola)
{
    Nodo_Cola *aux1;
    Nodo_Cola *aux2;
    aux1 = *Cola;
    if (aux1 != NULL)
    {
        aux2 = aux2->info;
        aux1 = aux1->sig;
        *Cola = aux1;
        TAM_COLA--;
        return aux2;
    }
    else
    {
        return NULL;
    }
}
```

```
Nodo_Cola *creanodo_cola(Nodo_BFS *nodo)
{
    Nodo_Cola *aux;
    aux = (Nodo_Cola *)malloc(sizeof(Nodo_Cola));
    aux->info = nodo;
    aux->sig = NULL;
    return (aux);
}
```

```
Nodo_BFS *set_hijos(Nodo_BFS *Hijos[fact_ram], Nodo_BFS *n)
{
    int i;
    n->hijos = Hijos;
    if (n->hijos != NULL)
    {
        for (i = 0; i < fact_ram; i++)
        {
            Hijos[i]->padre = n;
        }
    }
    return n;
}
```

```
void Encolar(Nodo_Cola **C, int elem, int b)
{
    Nodo_Cola *nuevo, *aux1, *aux2;
    aux1 = *C;
    nuevo = creanodo_cola(elem);
    while (aux1 != NULL)
    {
        aux2 = aux1;
        aux1 = aux1->sig;
    }
    if (*C == NULL)
    {
        *C = nuevo;
        if (b)
        {
            TAM_COLA++;
        }
    }
    else
    {
        aux2->sig = nuevo;
        if (b)
        {
            TAM_COLA++;
        }
    }
}
```

# REFERENCIAS


[1] "Top Programming Languages 2022", *IEEE Spectrum*, 23 de agosto de 2022. https://spectrum.ieee.org/top-programming-languages-2022 (accedido 28 de octubre de 2022).

[2] "Luda - Azcapotzalco". http://aniei.org.mx/paginas/uam/index.html.