

# Práctica 1

Aguilar Zúñiga,Gibran 308071087  
Alexis Hernández castro 313006636  
Jesus Martin Ortega Martinez 310183534  
Daniel Lopez Hernández 309167282  
Jaime Alberto Martínez López 309256753

10 de diciembre de 2018

## Resumen

Introducir al lenguaje C y manejo de apuntadores y memoria.

## 1. Preguntas

### NOTA:

Para ejecutar los programas con Makefile debes pararte en la carpeta adecuada y poner make para generar el ejecutable, despues ./Programas y se ejecutara el programa correcto. Ejemplo de ejecución con Makefile.

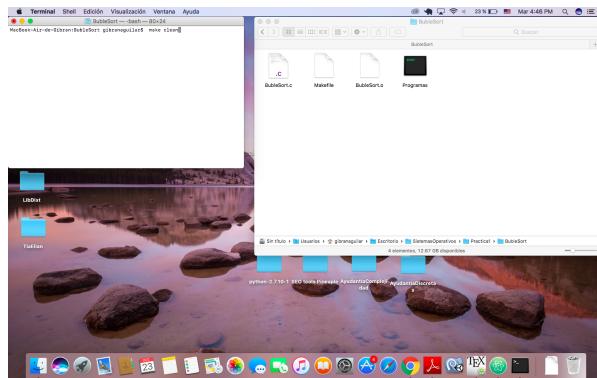


Figura 1: Ejecución con MakeFile

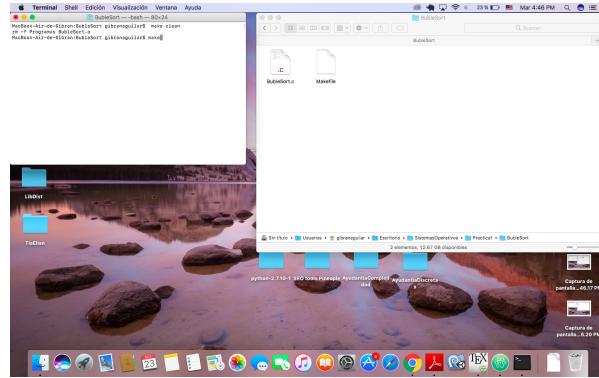


Figura 2: Ejecución con MakeFile

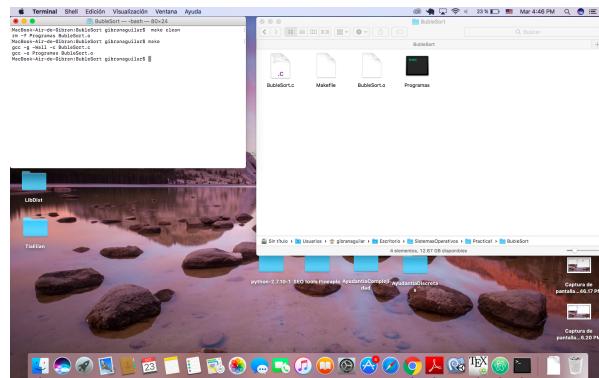


Figura 3: Ejecución con MakeFile

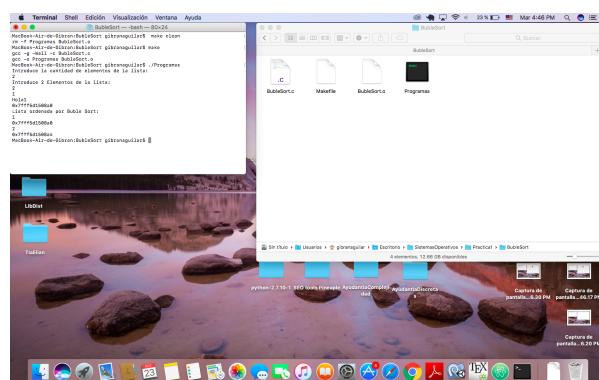
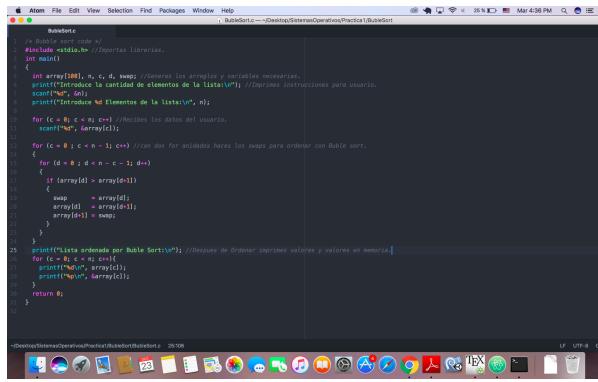


Figura 4: Ejecución con MakeFile

## 1. Ordenamiento burbuja con apuntadores.

- a) Realizar un programa en C, en el cual con cada iteración del ordenamiento burbuja, se modifiquen las referencias a los datos en memoria, y no los datos contenidos en la misma.



```
#include <stdio.h>
//Bubble sort code
//Entradas: n, c, d, swap //Entradas: Los arreglos y variables necesarias.
//Salida: n, c, d //Imprime los valores y direcciones de memoria.
int main()
{
    int array[100], n, c, d, swap; //Declara las variables necesarias.
    printf("Introduce la cantidad de elementos de la lista:\n");
    scanf("%d", &n);
    printf("Introduce los elementos de la lista:\n");
    for (c = 0; c < n; c++)
    {
        scanf("%d", &array[c]);
    }
    for (c = 0; c < n - 1; c++)
    {
        for (d = 0; d < n - c - 1; d++)
        {
            if (array[d] > array[d + 1])
            {
                swap = array[d];
                array[d] = array[d + 1];
                array[d + 1] = swap;
            }
        }
    }
    printf("Lista ordenada por Bubble Sort:\n");
    for (c = 0; c < n; c++)
    {
        printf("%d\n", array[c]);
    }
    return 0;
}
```

Figura 5: Código Buble Sort

- b) Se deber á imprimir en cada iteraci ón las direcciones de memoria empleando formateadores de cadena específicos para esta acción (%p)

## 2. Keylogger

- a) Empleando la llamada al sistema read en un sistema operativo derivado de UNIX, interceptar la informaci ón del dispositivo de entrada jtecladoj y guardarla en un archivo de texto.
- b) Las entradas del teclado se se almacenan en estructuras(structs) de tipo input event, que almacenan entre otros datos el c ódigo de la tecla pulsada o liberada, la definici ón de este tipo de estructuras s encuentra en sys/types.h y linux/input.h.
- c) El dispositivo que genera este tipo de estructuras se encuentra de manera l ógica en /dev/input/ en sistemas derivados de UNIX y se maneja como un evento, el evento cambia entre cada sistema operativo.

## 3. Multiplicaci ón dinamica de matrices.

- a) Empleando las funciones malloc(), calloc(), realloc() y free(), crear arreglos bidimensionales que representen matrices a multiplicar.

```

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    int m1=2; //filas de la matriz
    int n1=3; //columnas de la matriz
    int m2=3; //filas de la matriz
    int n2=2; //columnas de la matriz
    int i,j,k,l;
    int P[m1][n1];
    int Q[m2][n2];
    int R[m1][n2];

    printf("Número de renglones de la matriz P=%d\n",m1);
    printf("Número de columnas de la matriz P=%d\n",n1);
    scanf("%d", &m1);
    printf("Número de renglones de la matriz Q=0-%d\n",n2);
    printf("Número de columnas de la matriz Q=0-%d\n",m2);
    scanf("%d", &n2);
    //verificacion de dimensiones de matrices para saber que es posible multiplicar dichas matrices.
    if(m1!=n2)
        printf("No es posible hacer la multiplicación\n");
    system("pause");
    return 0;
}

//verifico que las matrices tengan espacio suficiente.
p = (int**) malloc(m1 * sizeof(int));
q = (int**) malloc(n2 * sizeof(int));
r = (int**) malloc(m1 * n2 * sizeof(int));
if(p==NULL)
{
    printf("Insuficiente Espacio de Memoria"); exit(-1);
}
if(q==NULL)
{
    printf("Insuficiente Espacio de Memoria"); exit(-1);
}
if(r==NULL)
{
    printf("Insuficiente Espacio de Memoria"); exit(-1);
}
//Inicializo el apuntador como arreglo de arreglos con el valor que proporciona el usuario.
for(i=0; i<m1; i++)
{
    p[i] = (int*) malloc(m1 * sizeof(int));
    for(j=0; j<n2; j++)
    {
        q[j] = (int*) malloc(n2 * sizeof(int));
        for(k=0; k<n1; k++)
        {
            r[i][j] = 0;
        }
    }
}
//Inicializo el apuntador como arreglo de arreglos con el valor que proporciona el usuario.
for(i=0; i<m1; i++)
{
    for(j=0; j<n1; j++)
    {
        p[i][j] = 1;
    }
}
//Inicializo el apuntador como arreglo de arreglos con el valor que proporciona el usuario.
for(i=0; i<n2; i++)
{
    for(j=0; j<n1; j++)
    {
        q[i][j] = 1;
    }
}
//Imprimiendo para cargar los valores.
for(i=0; i<m1; i++)
{
    for(j=0; j<n1; j++)
    {
        printf("Escribe el valor de la matriz P (%d,%d)->(%d,%d)\n", i+1, j+1);
        scanf ("%d", &p[i][j]);
    }
}

```

Figura 6: Código Matrices Multiplicación

- b) Las dimensiones de las matrices deberán ser especificadas por el usuario, y no se debe emplear mas espacio de almacenamiento que el requerido por las matrices.

```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    //verifico que las matrices tengan espacio suficiente.
    if(m1==NULL)
    {
        printf("Insuficiente Espacio de Memoria"); exit(-1);
    }
    //Inicializo el apuntador como arreglo de arreglos con el valor que proporciona el usuario.
    for(i=0; i<m1; i++)
    {
        p[i] = (int*) malloc(m1 * sizeof(int));
        for(j=0; j<n1; j++)
        {
            q[j] = (int*) malloc(n1 * sizeof(int));
            for(k=0; k<n2; k++)
            {
                r[i][j] = 0;
            }
        }
    }
    //Inicializo el apuntador como arreglo de arreglos con el valor que proporciona el usuario.
    for(i=0; i<n2; i++)
    {
        for(j=0; j<n1; j++)
        {
            p[i][j] = 1;
        }
    }
    //Inicializo el apuntador como arreglo de arreglos con el valor que proporciona el usuario.
    for(i=0; i<n1; i++)
    {
        for(j=0; j<n2; j++)
        {
            q[i][j] = 1;
        }
    }
    //Imprimiendo para cargar los valores.
    for(i=0; i<m1; i++)
    {
        for(j=0; j<n1; j++)
        {
            printf("Escribe el valor de la matriz P (%d,%d)->(%d,%d)\n", i+1, j+1);
            scanf ("%d", &p[i][j]);
        }
    }
}

```

Figura 7: Código Matrices Multiplicación

- c) Se tiene la opción de solicitar los elementos de las matrices al usuario o generarlos de manera automatica a través de funciones de generación de numeros aleatorios propias de C. En ambos casos de deben imprimir las matrices y el resultado de la multiplicación.

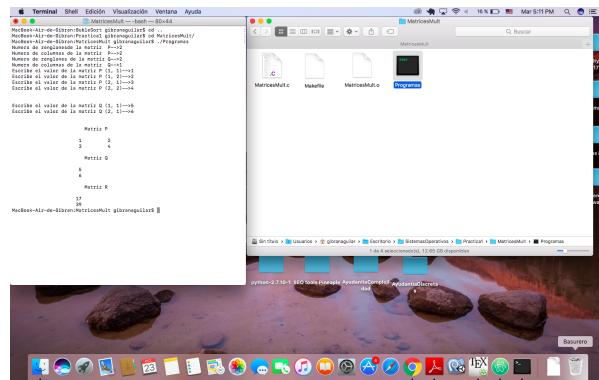


Figura 8: Código Matrices Multiplicación

#### 4. 2K38Y

- a) Imprimir en pantalla la fecha exacta con segundos para la Ciudad de México en la que la forma de medir el tiempo con epoch desbordará el espacio de almacenamiento valido para un entero de 32 bits.

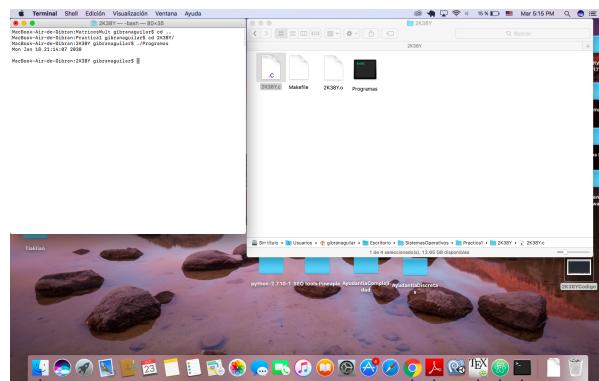
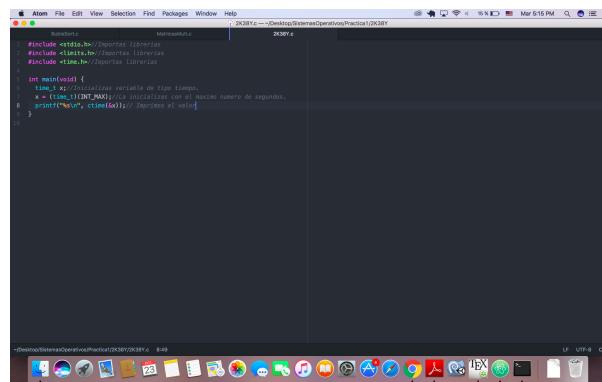


Figura 9: Ejecución 2K38Y

- b) Restricción. no más de 5 líneas de código dentro de la función main.



A screenshot of a terminal window titled "2K38Y.c" containing the following C code:

```
#include <stdio.h> // Importa la biblioteca de entrada/salida
#include <limits.h> // Importa las limitaciones de los tipos de datos
#include <time.h> // Importa las librerías de tiempo

int main(void) {
    time_t max; // Declara una variable de tipo tiempo.
    max = (long)INT_MAX; // Inicializa con el máximo número de segundos.
    printf("%d", max); // Imprime el valor
}
```

Figura 10: Código 2K38Y