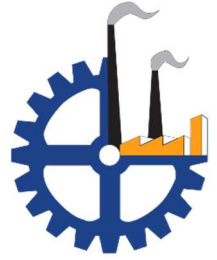




TECNOLÓGICO  
NACIONAL DE MÉXICO



# INSTITUTO TECNOLÓGICO DE CHIHUAHUA

## Arquitectura de Prog. Para Hardware

Laboratorio 2  
Memoria dinámica

Alumno:

David Sebastián Gutiérrez Ortega

18060734

Docente:

Chacón Aldama Alfredo

## Instrucciones

Realizar lo especificado en el Lab1 del tema de estructuras, utilizando arreglos dinámicos (validando su asignación de memoria en el heap)

## Objetivo

Utilizar el conocimiento adquirido sobre el tema de memoria dinámica para mejorar el programa realizado en el laboratorio 1.

## Metodología

Diseñe un programa en lenguaje C que utilice estructuras de datos, arreglos, apuntadores, funciones etc. para que clasifique microcontroladores según su tamaño de arquitectura (32, 16 y 8 bits). Además, se hará uso de la función malloc la cual nos permite crear un objeto de cualquier tipo, incluyendo tipos definidos por el usuario, y devuelve un puntero (del tipo adecuado) al objeto creado.

Por último, se imprimirán los datos capturados clasificados por su arquitectura

## Materiales

- Ide Dev C++
- Computadora
- Conocimientos básicos de C

## Desarrollo

Se declararon dos estructuras anidadas, así como los datos de entrada para el programa según las instrucciones dadas como se observa en la figura 1.

```
/*Declaracion de Estructura*/
typedef struct micros{
    char *nombre;
    char *fabricante;
    int archi;
}micros_e;

typedef struct{
    int n;
    micros_e *dato;
}registro;
```

Figura 1. Declaración de las estructuras

\*Ente caso las estructuras se declaran como si fueran un puntero\*

Dentro de la función “main” se declara la variable “x” de tipo “registro” en la cual se encuentra los datos que contienen las estructuras, también se hace la captura del número de microcontroladores el cual se almacena en “x.n”.

```
int main()
{
    int i=0;
    registro x;
    portada();

    printf("Numero de micros: ");
    scanf("%d",&x.n);

    //micros dato[n];
    x.dato = (micros_e*)malloc(x.n*sizeof(micros_e));
```

Figura 2. Función main

Se realiza la asignación de memoria de las variables que vamos a utilizar y el tamaño según dado el número de microcontroladores que se vayan a capturar como se observa en la figura 2.

Dentro de la función de la figura 3 se puede observar que se utiliza un ciclo “for” para hacer la captura de los datos.

```
x.dato = (micros_e*)malloc(x.n*sizeof(micros_e));
/*DATOS DE ENTRADA O CAPTURACION DE DATOS*/
for(i=0;i<x.n;i++){
    printf("\nNombre: ");
    scanf("%s",&x.dato[i].nombre);
    printf("\nFabricante: ");
    scanf("%s",&x.dato[i].fabricante);
    printf("\nArquitectura: ");
    scanf("%d",&x.dato[i].arqui);

    /*validacion para el tipo de arquitectura*/
    while (x.dato[i].arqui!=8 && x.dato[i].arqui!=16 &&x.dato[i].arqui!=32 )
    {
        printf("Valor de arquitectura no valida!\n");
        printf("Arquitectura:");
        scanf("%d",&x.dato[i].arqui);
    }
}
```

Figura 3. Capitulación de datos con validación de su arquitectura

El ciclo while, nos permite hacer la validación del tipo de arquitectura de los microcontroladores ingresados, utilizando el operando “&&”, esto significa que, si el valor de la arquitectura es diferente a 8,16 o 32, mostrará un mensaje de que la arquitectura ingresada es incorrecta, por lo que pedirá que se capture un dato que se encuentre dentro de las especificaciones del programa como se ve en la figura 3.

Y por último se utiliza un “for” para cada tipo de arquitectura como se puede observar en la figura 4.

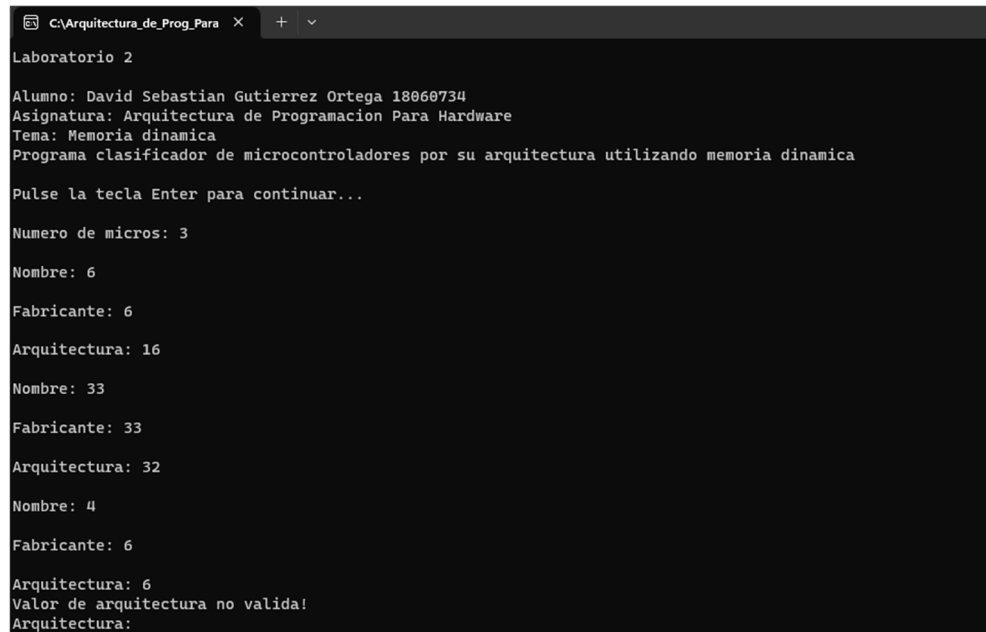
```
printf("\n**Microcontroladores de 8 bits**\n");
for(i=0;i<x.n;i++){
    if(x.dato[i].arqui == 8){
        printf("\tNombre: %s\n",8x.dato[i].nombre);
        printf("\tFabricante: %s\n",8x.dato[i].fabricante);
        printf("\tArquitectura: %d\n\n",x.dato[i].arqui);
    }
}

printf("\n**Microcontroladores de 16 bits**\n");
for(i=0;i<x.n;i++){
    if(x.dato[i].arqui == 16){
        printf("\tNombre: %s\n",8x.dato[i].nombre);
        printf("\tFabricante: %s\n",8x.dato[i].fabricante);
        printf("\tArquitectura: %d\n\n",x.dato[i].arqui);
    }
}

printf("\n**Microcontroladores de 32 bits**\n");
for(i=0;i<x.n;i++){
    if(x.dato[i].arqui == 32){
        printf("\tNombre: %s\n",8x.dato[i].nombre);
        printf("\tFabricante: %s\n",8x.dato[i].fabricante);
        printf("\tArquitectura: %d\n\n",x.dato[i].arqui);
    }
}
```

Figura 4. Impresión de los datos ingresados según su arquitectura.

## Resultados



```
C:\Arquitectura_de_Prog_Para  x  +  v
Laboratorio 2
Alumno: David Sebastian Gutierrez Ortega 18060734
Asignatura: Arquitectura de Programacion Para Hardware
Tema: Memoria dinamica
Programa clasificador de microcontroladores por su arquitectura utilizando memoria dinamica
Pulse la tecla Enter para continuar...
Numero de micros: 3
Nombre: 6
Fabricante: 6
Arquitectura: 16
Nombre: 33
Fabricante: 33
Arquitectura: 32
Nombre: 4
Fabricante: 6
Arquitectura: 6
Valor de arquitectura no valida!
Arquitectura:
```

Figura 5. Portada del programa y validación de la arquitectura.

\*En este caso muestra un mensaje y pide introducir una arquitectura correcta\*

```
C:\Arquitectura_de_Prog_Para X + v

**Microcontroladores de 8 bits**
    Nombre: 8
    Fabricante: 8
    Arquitectura: 8

    Nombre: 8
    Fabricante: 8
    Arquitectura: 8

    Nombre: 8
    Fabricante: 8
    Arquitectura: 8

**Microcontroladores de 16 bits**
    Nombre: 8
    Fabricante: 8
    Arquitectura: 16

**Microcontroladores de 32 bits**
    Nombre: 16
    Fabricante: 16
    Arquitectura: 32

    Nombre: 32
    Fabricante: 8
    Arquitectura: 32
```

Figura 6. Impresión de los datos

## Conclusión

Se puede concluir que la memoria dinámica nos permite modificar el valor o la dimensión de los arreglos o capacidad de una variable, en comparación de cuando son declaradas con un tamaño, esta no puede ser modificada cuando uno lo desee. Sin embargo, también puede ocurrir casos en los que se corrompan los datos, aunque no genere algún error al compilar.

Utilizar memoria dinámica nos permite que los datos existan hasta que explícitamente sean liberados usando free, además se puede modificar el tamaño de memoria asignada en tiempo de ejecución sin problema alguno.

## Referencias

Libro de la unidad I en la plataforma SEL

## Anexos

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
//Declaracion de variables globales
```

```
//int i;
```

```
/*Declaracion de Estructura*/
```

```
typedef struct micros{
```

```
    char *nombre;
```

```
    char *fabricante;
```

```
    int archi;
```

```
}micros_e;
```

```
typedef struct{
```

```
    int n;
```

```
    micros_e *dato;
```

```
}registro;
```

```
/*Prototipo de funcion*/
```

```
//void microsPrint(struct micros dato[]);
```

```
//Declaracion de Estructura
```

```
/*typedef struct micros_e{
```

```
    char nombre[20];
```

```
    char fabricante [20];
```

```
    int archi;
```

```
}micros;*/
```

```
//Prototipo de funcion
```

```

void portada(){
    printf("\n\nLaboratorio 2 \n\n");
    printf("Alumno: David Sebastian Gutierrez Ortega 18060734\n");
    printf("Asignatura: Arquitectura de Programacion Para Hardware\n");
    printf("Tema: Memoria dinamica\n");
    printf("Programa clasificador de microcontroladores por su arquitectura utilizando memoria
dinamica\n\n");

    printf("Pulse la tecla Enter para continuar...\n");
    getchar();
}

```

```

/*void impresion_datos(struct micros_e dato[]){
    printf("\tNombre: %s\n",dato[i].nombre);
    printf("\tFabricante: %s\n",dato[i].fabricante);
    printf("\tArquitectura: %d\n\n",dato[i].arqui);
}

```

```

void portada();
void impresion_datos(struct micros_e dato[]);*/

```

```

void portada();

```

```

int main()
{
    int i=0;
    registro x;
    portada();

    printf("Numero de micros: ");
    scanf("%d",&x.n);

    //micros dato[n];

```

```

x.dato = (micros_e*)malloc(x.n*sizeof(micros_e));
/*DATOS DE ENTRADA O CAPTURACION DE DATOS*/
for(i=0;i<x.n;i++){
    printf("\nNombre: ");
    scanf("%s",&x.dato[i].nombre);
    printf("\nFabricante: ");
    scanf("%s",&x.dato[i].fabricante);
    printf("\nArquitectura: ");
    scanf("%d",&x.dato[i].arqui);

    /*validacion para el tipo de arquitectura*/
    while (x.dato[i].arqui!=8 && x.dato[i].arqui!=16 &&x.dato[i].arqui!=32 )
    {
        printf("Valor de arquitectura no valida!\n");
        printf("Arquitectura:");
        scanf("%d",&x.dato[i].arqui);
    }
}

//Datos(dato);
// void Datos(struct micros_e dato[]){
// int i;

system("cls");/*Limpia consola*/

printf("\n**Microcontroladores de 8 bits**\n");

for(i=0;i<x.n;i++){
    if(x.dato[i].arqui == 8){
        printf("\tNombre: %s\n",&x.dato[i].nombre);
        printf("\tFabricante: %s\n",&x.dato[i].fabricante);
        printf("\tArquitectura: %d\n\n",x.dato[i].arqui);
    }
}

```



```

    }
}

printf("\n**Microcontroladores de 16 bits**\n");

for(i=0;i<x.n;i++){
    if(x.dato[i].arqui == 16){
        printf("\tNombre: %s\n",&x.dato[i].nombre);
        printf("\tFabricante: %s\n",&x.dato[i].fabricante);
        printf("\tArquitectura: %d\n\n",x.dato[i].arqui);
    }
}

printf("\n**Microcontroladores de 32 bits**\n");

for(i=0;i<x.n;i++){
    if(x.dato[i].arqui == 32){
        printf("\tNombre: %s\n",&x.dato[i].nombre);
        printf("\tFabricante: %s\n",&x.dato[i].fabricante);
        printf("\tArquitectura: %d\n\n",x.dato[i].arqui);
    }
}
}

```