Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño F.I.A.D.

Campus Ensenada

Ramón Mejía Martínez 372099

Grupo: 932

Actividad 9

Programación Estructurada

Ensenada, Baja California a 30 de septiembre de 2023.

Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Ramón Mejía Martínez

Matrícula: 372099

Maestro: Pedro Núñez Yépiz

Actividad No. : 10

Tema - Unidad : Structs.

Ensenada Baja California a 15 de octubre del 2023.

1. Introducción

En esta actividad se usarán los structs, un "struct" permite a los programadores combinar diferentes tipos de datos bajo una única estructura personalizada, lo que facilita la creación de registros.

2. Competencia

Esta actividad tiene como objetivo explorar el uso de "structs" en C y entender cómo pueden mejorar la organización y manipulación de datos en programas.

3. Fundamentos

Los structs son un tipo de dato que permite combinar diferentes tipos de datos en una única estructura personalizada. Son muy útiles para el manejo de distintos datos relacionados entre sí.

Definir structs:

```
struct{
        tipo_dato variable1;
        tipo_dato variable2;.
        .
        .
        .
};
```

Estas estructuras se pueden renombrar usando typedef:

```
typedef struct{
        tipo_dato campo1;
        tipo_dato campo2;.
        .
        .
        .
}nombre;
```

Para acceder a cada campo se usa un punto entre el nombre de la struct y el campo al que se quiere acceder, por ejemplo:

```
        nombreStruct.campo1;
```

Se pueden crear arreglos de structs.

4. Procedimiento

1.

ACTIVIDAD 10

FUNCIONES y METODOS DE ORDENACION Y BUSQUEDA ESTRUCTURAS Y LIBRERIAS

INSTRUCCIONES:
1.- Realiza un programa en C que utilice una librería propia
(Funciones de validar numeros y , cadenas)
2.- Realiza reporte de práctica
3.- Sube a Blackboard, programa, librería, y reporte de practica y PDF anexo con capturas y código

---

2.

ACTIVIDAD 10

REALICE EL SIGUIENTE PROGRAMA QUE CONTENGA UN MENÚ.

MENÚ
1.- AGREGAR (AUTOM 10 REGISTROS)
2.- AGREGAR MANUAL
3- ELIMINAR REGISTRO (lógico)
4.- BUSCAR
5- ORDENAR
6.- IMPRIMIR
0.- SALIR

UTILIZAR UN ARREGLO DE 500 REGISTROS
SE DEBERÁ **UTILIZAR ESTRUCTURAS** CON LOS DATOS BÁSICOS DE UN ALUMNO (status, Matricula, ApPat, ApMat, Nombre, Edad, Sexo)
Busqueda y Ordenacion por campo MATRICULA

5. Resultados y conclusiones:

Opción 1:

En el menú llamamos a la función randAlum 10 veces, cada llamada se incrementa el contador i que representa el número de alumnos.

```
case 1:
    printf("\nSE GENERARON 10 ALUMNOS\n");
    for(j=0;j<10;j++)
    {
        randAlum(alumnos,i++);
    }
    printf("%d",i);
    break;
```

Función randAlum:

- Generamos el estado, sexo, edad de forma aleatoria usando rand.
- Llamamos a la función genMatri para asignar el valor generado a la matricula.
- Dependiendo del sexo generado se le asigna un nombre correspondiente.

```
void randAlum(StructAlum alumnos[],int i)
{
    int prob;   //se usa para controlar la probabilidad de tener 1 o 2 apellidos
    alumnos[i].status=rand()%2;
    alumnos[i].matricula=genMatri(alumnos,i);
    alumnos[i].sexo=(rand()%2)+1;
    alumnos[i].edad=(rand()%150)+1;
    if(alumnos[i].sexo==1)
    {
        genH_Name(alumnos,i);
    }
    else
    {
        genM_Name(alumnos,i);
    }
```

Función genMatri:

Genera un numero aleatorio y si no lo devuelve, así generando una matricula que no este en el vector.

```
int genMatri(StructAlum alumnos[],int i)
{
    int num;
    do
    {
        num=(rand()%100000)+300000;
    }while(buscarAlum(alumnos,i,num)!=-1);
    return num;
}
```

Función genH_Name y genM_Name:

- Estas funciones hacen exactamente lo mismo con la única diferencia que buscan en el vector m_name para nombres de mujer y h_name para nombres de hombre.
- El if controla con %50 de probabilidad si tendrá 1 o 2 nombres.
- Si tiene un nombre copia el nombre.
- Si tiene dos nombres copia el primer nombre y concatena un segundo nombre usando la función strConcat.

```
void genH_Name(StructAlum alumnos[],int i)
{
    if(rand()%2)
    {
        strcopy(alumnos[i].nombre,m_name[rand()%28]);
    }
    else
    {
        strcopy(alumnos[i].nombre,m_name[rand()%28]);
        strConcat(alumnos[i].nombre,m_name[rand()%28]);
    }
}
```

Función strConcat:

- Usamos la función strlen para calulcar la longitud de la cadena principal.
- Reemplaza el carácter nulo de la cadena principal por un espacio.
- Dentro del while asigna la cadena a concatenar después del espacio de la segunda.
- Asigna el carácter nulo después de haber concatenado.

```
void strConcat(char str2[],char str1[])
{
    int j,k=0;
    j=strlen(str2); //Calcula la longitud de la cadena principal
    str2[j++]=' ';  //Cambia el caracter nulo por espacio
    while(str1[k]!='\0')    //Se repite hasta que la cadena que se concatenara llegue al caracter nulo
    {
        str2[j]=str1[k];    //Despues del espacio se le asigna la segunda cadena a la primera
        k++;
        j++;
    }
    str2[j]='\0';   //Se asigna el caracter nulo despues de haber concatenado
}
```

Función randAlum:

- Para la asignación de apellidos se usa un switch para determinar la probabilidad de que el alumno generado tenga 1 o 2 apellidos.
- Copia en la casilla de apellido correspondiente un apellido de la librería aleatorio.

```
prob=rand()%10;
switch(prob)
{
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
        strcopy(alumnos[i].apPat,ap[rand()%48]);
        strcopy(alumnos[i].apMat,ap[rand()%48]);
        break;
    case 7:
    case 8:
    case 9:
        strcopy(alumnos[i].apPat,ap[rand()%48]);
        strcopy(alumnos[i].apMat,"  ");
        break;
}
```

Opción 2:

Se llama a la función scanAlum y se incrementa el contador de alumnos al salir de la función.

```
    case 2:
        printf("AGREGAR UN ALUMNO\n\n");
        scanAlum(alumnos,i++);
        break;
```

Función scanAlum:

- Leemos y validamos por separado todas las características del alumno usando las funciones de nuestra librería.
- Usamos un dowhile para buscar si la matricula ya existe o no usando la función buscarAlumno.
- Convierte las cadenas leídas a mayúsculas.

```
void scanAlum(StructAlum alumnos[],int i)
{
    int matri,validmatri;
    alumnos[i].status=validNum("Ingresa su status (0 o 1): ",0,1);
    do
    {
        matri=validNum("Ingresa su matricula: ",300000,399999);
        validmatri=buscarAlum(alumnos,i,matri);
        if(validmatri!=-1)
        {
            printf("\nEsa matricula ya existe\n");
        }
    }while(validmatri!=-1);
    alumnos[i].matricula=matri;
    printf("\nIngresa su apellido paterno: ");
    matri=validStr(alumnos[i].apPat);
    strM(alumnos[i].apPat);
    printf("\nIngresa su apellido materno: ");
    matri=validStr(alumnos[i].apMat);
    strM(alumnos[i].apMat);
    printf("\nIngresa su nombre: ");
    matri=validStr(alumnos[i].nombre);
    strM(alumnos[i].nombre);
    alumnos[i].edad=validNum("\nIngresa su edad ",1,150);
    alumnos[i].sexo=validNum("\nIngresa su sexo\n1) Hombre\n2) Mujer\nEscoge una opcion: ",1,2);
}
```

Opción 3:

- Lee la matricula que desea eliminar.
- Si la encuentra muestra los datos del alumno y llama a la función delAlum decrementando el contador de alumnos después.
- Si no la encuentra solo muestra un mensaje y regresa al menú.

```
case 3:
    printf("ELIMINAR ALUMNO\n\n");
    del=buscarAlum(alumnos,i,validNum("INGRESA LA MATRICULA DEL ALUMNO: ",300000,399999));
    if(del!=-1)
    {
        printf("\nSE ELIMINO EL ALUMNO: \n");
        printf("| No  | Estatus | Matricula | Apellido Paterno | Apellido Materno | Nombre
        printAlum(alumnos,del);
        delAlum(alumnos,i--,del);
    }
    else
    {
        printf("\nESE ALUMNO NO EXISTE\n");
    }
    break;
```

Función delAlum:

- Recorremos el vector de alumnos desde el alumno que se desea eliminar.
- La posición del alumno actual es remplazada por el siguiente alumno, de esta forma solo se pierde la información del primer alumno.

```
void delAlum(StructAlum alumnos[],int i,int pos)
{
    int j;
    for(j=pos;j<i;j++)     //Recorre las posiciones de los alumnos
    {
        alumnos[j]=alumnos[j+1];   //La posición del alumno actual es remplazada por el siguiente alumno
    }
}
```

Opción 4:

- Leemos y validamos la matricual que se desea buscar.
- Buscamos la posición del alumno llamando a la función buscarAlum, si la encuentra muestra sus datos, si no muestra un mensaje y regresa al menú.

```
case 4:
    printf("BUSCAR ALUMNO\n\n");
    pos=buscarAlum(alumnos,i,validNum("INGRESA LA MATRICULA DEL ALUMNO: ",300000,399999));
    if(pos!=-1)
    {
        printf("| No  | Estatus | Matricula | Apellido Paterno | Apellido Materno | Nombre
        printAlum(alumnos,pos);
    }
    else
    {
        printf("\nESE ALUMNO NO EXISTE\n");
    }
    break;
```

Función buscarAlum:

- Recorre el vector de alumnos para comparar la matricula buscada con alguna ya existente.
- Si la encuentra devuelve la posición, si no devuelve -1.

```
int buscarAlum(StructAlum alumnos[],int i,int num)
{
    int j;
    for(j=0;j<i;j++)    //Recorre el vector de alumnos
    {
        if(num==alumnos[j].matricula)   //Compara la matricula buscada con las existentes
        {
            return j;   //Si la encuentra regresa la posicion
        }
    }
    return -1;  //Si no la encontro regresa -1
}
```

Opción 5:

- Si no hay alumnos o hay uno solo muestra un mensaje y regresa al menú.
- Si hay mas de 1 llama a la función orndenarAlum.

```
case 5:
    printf("SE ORDENARON LOS ALUMNOS POR MATRICUAL\n\n");
    switch(i)
    {
        case 0:
            printf("\nAUN NO HAY ALUMNOS\n");
            break;
        case 1:
            printf("\nSOLO HAY UN ALUMNO\n");
            break;
        default:
            ordenarAlum(alumnos,i);
    }
    break;
```

Función orndenarAlum:

Se usa el método de la burbuja mejorado para ordenar según las matrículas de los alumnos.

```
void ordenarAlum(StructAlum alumnos[],int n)
{
    int i,j,swap;
    StructAlum temp;   //guarda la posicion del alumno para intercambiarla con otro
    for(i=0;i<n-1;i++)
    {
        swap=0; //Se asigna 0 a la bandera de intercambios
        for(j=0;j<n-i-1;j++)
        {
            if(alumnos[j].matricula>alumnos[j+1].matricula) //Si la matricula siguiente es menor a la actual se
            {
                temp=alumnos[j];    //guarda los datos del alumno actual
                alumnos[j]=alumnos[j+1];    //asigna los datos del siguiente alumno al actual
                alumnos[j+1]=temp;  //asgina los datos guardados del primer alumno al segundo alumno alumno
                swap=1; //asigna 1 a la bandera de intercambios
            }
        }
        if (swap==0)
        {
            i=n;    //Si no se hicieron intercambios i se iguala a n para que termine el ciclo
        }
    }
}
```

Opción 6:

- Si no hay alumnos muestra un mensaje y regresa al menú.
- Si hay alumnos recorre todas las posiciones con un for hasta el numero de alumnos e imprime llamando a la función printAlum.

```
case 6:
    if(i>0)
    {
        printf("| No  | Estatus | Matricula | Apellido Paterno | Apellido Materno | Nombre
        for(j=0;j<i;j++)
        {
            printAlum(alumnos,j);
        }
    }
    else
    {
        printf("\nAUN NO HAY ALUMNOS\n");
    }
    break;
```

Función printAlum:

- Recibe el índice del alumno que se imprimirá e imprime todos sus datos.
- Usa un if para comprobar el sexo y escribir el correspondiente.

```
void printAlum(StructAlum alumnos[],int j)
{
    printf("| %3d |    %d    | %9d | %-16s | %-17s| %-19s | %4d |",j+1,alumnos[j].status,alumnos[j].matricula,alu
    if(alumnos[j].sexo==1)
    {
        printf(" Hombre |\n");
    }
    else
    {
        printf(" Mujer  |\n");
    }
}
```

Conclusión:

Las structs facilitaron el manejo de datos, al realizar muchas opciones del menú se imprimen mas de una vez los alumnos alguna de esas aun desordenados, esto solo sucede al compilar en visual studio code, se imprime normalmente compilando en devc/c++, esto no afecta a la navegación por el programa, se puede manipular correctamente los registros de alumnos.

6. Anexos

PDF:

https://drive.google.com/file/d/1ZVohS3xqWsQSJCzrJGlLx5yMHZPb0gym/view?usp=sharing

Repositorio:

https://github.com/Muners24/Programacion-Estructurada

7. REFERENCIAS

Diseño de algoritmos y su codificación en lenguaje C

Corona, M.A. y Ancona, M.A. (2011)..

España: McGraw-Hill.

ISBN: 9786071505712

Programación estructurada a fondo:implementación de algoritmos en C

:Pearson Educación.Sznajdleder, P. A. (2017)..

Buenos Aires,Argentina: Alfaomega

Como programar en C/C++

H.M. Deitel/ P.J. Deitel

Segunda edición

Editorial: Prentice Hall.

ISBN:9688804711

Programación en C.Metodología, estructura de datos y objetos

Joyanes, L. y Zahonero, I. (2001)..

España:McGraw-Hill.

ISBN: 8448130138

De Computadores, P., & Olave, T. A. (s/f). *Algoritmos de Búsqueda y Ordenamiento*.

Utfsm.cl. Recuperado el 9 de octubre de 2023, de https://www.inf.utfsm.cl/~noell/IWI-131-p1/Tema8b.pdf