Principios de Programación

El lenguaje C

1. Estructuras

Las estructuras son colecciones de variables relacionadas bajo un nombre. Las estructuras pueden contener variables de muchos tipos diferentes de datos - a diferencia de los arreglos que contienen unicamente elementos de un mismo tipo de datos.

1.1. Definicion de estructuras

Las estructuras son *tipos de datos derivados* - están construidas utilizando objetos de otros tipos. Considere la siguiente definición de estructura:

```
struct ejemplo {
        char c;
        int i;};
```

La palabra reservada **struct** indica se está definiendo una estructura. El identificador ejemplo es el nombre de la estructura. Las variables declaradas dentro de las llaves de la definición de estructura son los *miembros* de la estructura. Los miembros de la misma estructura deben tener nombres únicos mientras que dos estructuras diferentes pueden tener miembros con el mismo nombre. Cada definición de estructura debe terminar con un punto y coma.

La definición de struct ejemplo contiene un miembro de tipo char y otro de tipo int. Los miembros de una estructura pueden ser variables de los tipos de datos básicos (int, char, float,etc) o agregados como ser arreglos y otras estructuras. Una estructura no puede contener una instancia de si misma.

Declaramos variables del tipo estructura del siguiente modo:

```
struct ejemplo e1, a[10];
```

o alternativamente sin usar la palabra struct:

```
ejemplo e1, a[10];
```

Las declaraciones anteriores declaran variables e1 de tipo ejemplo y a de tipo arreglo de ejemplo de dimensión 10.

Se pueden declarar variables de tipo estructura ejemplo colocando sus nombres a continuación de la llave de cierre de la definición de estructura y el punto y coma, en el caso anterior:

```
struct ejemplo {
        char c;
        int i;} e1, a[10];
```

Una operación válida entre estructuras es asignar variables de estructura a variables de estructura del mismo tipo. Las estructuras no pueden compararse entre si.

Ejemplo

Consideremos la información de una fecha. Una fecha consiste de: el día, el mes, el año y posiblemente el día en el año y el nombre del mes. Declaramos toda esa información en una estructura del siguiente modo:

```
struct fecha {
        int dia;
        int mes;
        int anio;
        int dia_del_anio;
        char nombre_mes[9];
};
```

Un ejemplo de estructura que incluye otras estructuras es la siguiente estructura persona que incluye la estructura fecha:

```
struct persona {
        char nombre[tamanio_nombre];
        char direccion[tamanio_dir];
        long codigo_postal;
        long seguridad_social;
        double salario;
        fecha cumpleanios;
        fecha contrato;
};
```

1.2. Cómo inicializar estructuras

Las estructuras pueden ser inicializadas mediante listas de inicialización como con los arreglos. Para inicializar una estructura escriba en la declaración de la variable a continuación del nombre de la variable un signo igual con los inicializadores entre llaves y separados por coma por ejemplo:

```
ejemplo e1 = { 'a', 10 };
```

Si en la lista aparecen menos inicializadores que en la estructura los miembros restantes son automaticamente inicializados a 0.

Las variables de estructura también pueden ser inicializadas en enunciados de asignación asignandoles una variable del mismo tipo o asignandole valores a los miembros individuales de la estructura.

Podemos inicializar una variable del tipo fecha como sigue:

```
struct fecha f = {4, 7, 1776, 186, "Julio"};
```

1.3. Cómo tener acceso a los miembros de estructuras

Para tener acceso a miembros de estructuras utilizamos el operador punto. El operador punto se utiliza colocando el nombre de la variable de tipo estructura seguido de un punto y seguido del nombre del miembro de la estructura. Por ejemplo, para imprimir el miembro c de tipo char de la estructura e1 utilizamos el enunciado:

printf (" %c, e1.c);

Para acceder al miembro i de la estructura e1 escribimos: e1.i

En general, un miembro de una estructura particular es referenciada por una construcción de la forma:

nombre_de_estructura.miembro

por ejemplo para chequear el nombre de mes podemos utilizar:

if (strcmp(d.nombre_mes,"Agosto")==0) .....

1.4. Cómo utilizar estructuras con funciones

Las estructuras pueden ser pasadas a funciones pasando miembros de estructura individuales o pasando toda la estructura.

Cuando se pasan estructuras o miembros individuales de estructura a una función se pasan por llamada por valor. Para pasar una estructura en llamada por referencia tenemos que colocar el '*' o '&'.

Los arreglos de estructura como todos los demás arreglos son automaticamente pasados en llamadas por referencia.

Si quisieramos pasar un arreglo en llamada por valor, podemos definir una estructura con único miembro el array.

Una función puede devolver una estructura como valor.

Ejemplo

Consideraremos el ejemplo de un punto dado por dos coordenadas enteras.

```
struct punto {
        int x;
        int y;};

/* creo_punto: crea un punto a partir de sus coordenadas */
punto creo_punto(int a, int b)
{
        punto temp;

        temp.x=a;
        temp.y=b;
        return temp;
}
```

```
/* sumo_puntos: suma dos puntos */
punto sumo_puntos(punto p1,punto p2)
{
        p1.x += p2.x;
        p1.y += p2.y;
        return p1;
}

/* imprimo_punto: imprime las coordenadas de un punto */
void imprimo_punto(punto p)
{
      printf("Coordenadas del punto: %d y %d \n", p.x, p.y);
}
```

1.5. Typedef

La palabra reservada **typedef** proporciona un mecanismo para la creación de sinónimos (o alias) para tipos de datos anteriormente definidos. Por ejemplo:

```
typedef struct ejemplo Ejemplo;
```

define Ejemplo como un sinónimo de ejemplo.
Una forma alternativa de definir una estrutura es:

```
typedef struct {
        char c;
        int i;} Ejemplo;
```

Podemos ahora utilizar Ejemplo para declarar variables del tipo struct, por ejemplo

```
Ejemplo a[10];
```

typedef se utiliza a menudo para crear seudónimos para los tipos de datos básicos. Si tenemos por ejemplo un programa que requiere enteros de 4 bytes podría usar el tipo **int** en un programa y el tipo **long** en otro. Para garantizar portabilidad podemos utilizar typedef para crear un alias de los enteros de 4 bytes en ambos sistemas.

1.6. Ejemplo: simulación de barajar y distribuir cartas

El programa que sigue se basa en la simulación de barajar y distribuir cartas. El programa representa el mazo de cartas como un arreglo de estructuras, donde cada estructura contiene el número de la carta y el palo.

Las funciones son: **inicializar_mazo** que inicializa un array de cartas con los valores de las cartas ordenado del 1 al 12 de cada uno de los palos, **barajar** recibe un array de 48 cartas y para cada una de ellas se toma un número al

azar entre el 0 y el 47. A continuación se intercambia la carta original con la seleccionada al azar. Finalmente la función imprimir imprime las cartas. Se utiliza para imprimir las cartas luego de barajar.

La función copiar es auxiliar dentro de inicializar_mazo, es necesaria porque ISO C++ prohibe la asignación de arrays (ISO=International Organization for Standardization, red formada por distintos paises, no gubernamental. En algunos paises sus miembros son parte de la estructura gubernamental y en otros son parte del sector privado. En Uruguay depende del gobierno desde el 97).

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

struct carta {
        int numero;
        char palo[7];};

typedef carta Carta;
typedef char Palo[7];

void inicializar_mazo(Carta m[],Palo p[]);

void barajar(Carta m[]);

void imprimo(Carta m[]);

main ()
{
       Carta mazo[48];
       Palo p[4] = {"copa","oro","espada","basto"};

       srand(time(NULL));
       inicializar_mazo(mazo,p);
       barajar(mazo);
       imprimir(mazo);
       system("PAUSE");
}

void copiar(char a, char b, int largo)
{
      int i;

      for (i=0;i < largo;i++)
          a[i]=b[i];
}

void inicializar_mazo(Carta m[],Palo p[])
{
      int i;
```

```
        for (i=0; i < 48;i++)
        {
            m[i].numero=(i % 12)+1;
            copiar(m[i].palo,p[i/12],7);
        }
}

void barajar(Carta m[])
{
        int i,j;
        Carta temp;

        for (i=0; i < 48; i++)
        {
            j = rand() % 48;
            temp = m[i];
            m[i] = m[j];
            m[j] = temp;
        }
}

void imprimir(Carta m[])
{
        int i,j;
        char c;
        for (i=0; i < 48; i++)
        {
            printf (" %i de ", m[i].numero);
            printf (" %s ", m[i].palo);
            printf ("\n");
        }
}
```