Laboratorio 13

Estructuras

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas estructuras del lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando estructuras del lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Introducción

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 únicamente elementos de un mismo tipo de datos.

3.2. Sintaxis de estructuras

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

```
struct persona
	{
		char inicial;
		int edad;
	};
```

La palabra reservada struct indica que se está definiendo una estructura. El identificador "persona" 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 "persona" 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 sí misma.

Declaramos variables del tipo estructura del siguiente modo:

```
struct persona p1, e[10];
```

o alternativamente sin usar la palabra struct:

```
persona p1, e[10]
```

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

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

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];
} f0;
```

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

```
struct persona
       {
              char  inicial;
              int   edad;
              fecha cumpleanios;
              fecha contrato;
              };
```

3.3. Como 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

```
f0 = {20,6,2022};
```

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

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

Podemos inicializar una variable del tipo fecha como sigue:

```
fecha f1 = {31,6,2022,100,'Jun'};

struct fecha f1 = {31,6,2022,100,'Jun'};
```

3.4. 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 persona de tipo char de la estructura e1 utilizamos el enunciado:

```
printf(p1.nombre);
```

Ejemplo

```
#include "stdafx.h"
#include <iostream>
using namespace std;


int main()
{

	struct fecha {
		int dia;
		int mes;
		int anio;
		int dia_del_anio;
		char nombre_mes[9];
	} f0;

	struct persona
	{
		char nombre[10];
		char  inicial;
		int   edad;
		float nota;

	};

	f0 = {20,6,2022,100};
	fecha f1 = {31,10,2020,100,'Jun'};
	struct persona p1 = {"Juan",'J',19,18};
	cout << "El mes con f0 es " << f0.mes << endl;
	cout << "El mes con f1 es " << f1.mes << endl;
	cout << "El nombre p1 es " << p1.nombre << endl;
	cout << p1.nombre << endl;
	system("pause");
	return 0;
}
```

3.5. Como 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 automáticamente pasados en llamadas por referencia.

Si quisiéramos 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.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

struct punto {
	int x;
	int y;
};

punto creo_punto(int, int);
punto sumo_puntos(punto, punto);
void imprimo_punto(punto);

int main()
{
	sumo_puntos(creo_punto(1, 2), creo_punto(3, 4));
	system("pause");
	return 0;
}

/* 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 nn", p.x, p.y);
    }
```

3.6. 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 carta Carta;
```

define Carta como un sinónimo de ejemplo.

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.

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.

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


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

typedef carta Carta;
typedef char Palo[7];
void inicializarMazo(Carta m[], Palo p[]);
void barajar(Carta m[]);
void imprimir(Carta m[]);
```

```
void main()
{
	Carta mazo[48];
	Palo p[4] = { "\copa","\oro", "\espada","\basto" };
	srand(time(NULL));
	inicializarMazo(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 inicializarMazo(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");
	}
   }
```

1. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Implementar un programa que maneje un arreglo de estructuras para almacenar los nombres y las fechas de cumpleaños de sus n compañeros. Y debe mostrarse por pantalla quienes cumplen años en este mes.
2. Implementar un programa que maneje un arreglo de estructuras que calcule la nota final del Ciencia de la Computación. El programa debe permitir el ingreso de cualquier cantidad de alumnos y para cada alumno, se podrá ingresar nombre, grupo, nota de la primera fase, segunda fase, tercera fase y proyecto final. El porcentaje de cada ítem es 15%, 20%, 25% y 40% respectivamente.
3. Implemente un programa que maneje un arreglo de estructuras que solicite el nombre, edad y talla de 10 jugadores, debe mostrar por pantalla los que son menores de 20 años y tienen una talla mayor a 1,70 mts de altura.
4. Implemente un programa que maneje un arreglo de estructuras que solicite nombre, sexo y sueldo de los empleados de una empresa y debe mostrar por pantalla el menor y mayor sueldo.

2. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB13_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB13_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB13_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB13_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.