Algoritmos y Estructuras de Datos I

Trabajo Práctico N°6	Unidad 6
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo grupal
Metodología de Desarrollo: acordar	**Metodología de Corrección:** acordar docente
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

MARCO TEÓRICO

1. ¿Describir el Concepto de Estructura?

La estructura es un grupo de elementos de datos llamados miembros, que pueden ser de diferentes tipos y agruparse bajo una misma denominación. Esta estructura permite almacenar diferentes tipos de datos relacionados, lo cual ofrece mayor flexibilidad que los arrays, que sólo pueden almacenar datos de un mismo tipo. Dicho de otra manera, la estructura permite manejar la información como una unidad, lo que facilita trabajar con datos relacionados, donde cada campo puede tener un tipo distinto. Las estructuras también pueden contener otras estructuras, lo que brinda una gran flexibilidad para manejar datos complejos

2. Describir el concepto de Registro

Un registro es un tipo de datos estructurado, heterogéneo, lo que significa que cada uno de los componentes de un registro puede ser de un tipo de datos diferente. Cada componente de un registro se denomina campo del registro, y cada campo recibe un nombre conocido como nombre de campo.
En C++, los registros se denominan estructuras (struct). Los campos de un registro se llaman miembros de la estructura, y cada miembro tiene un nombre. Los registros permiten agrupar varios datos heterogéneos bajo un mismo nombre y acceder a sus elementos a través del nombre del campo utilizando el operador de acceso .

3. ¿Cómo se define una estructura?

Una estructura se define como un tipo de datos estructurados en el que cada valor es una colección de componentes. Estos componentes se agrupan bajo un único nombre, pero es posible acceder a cada uno de ellos individualmente. En C++, las estructuras (structs) permiten agrupar datos heterogéneos, es decir, elementos de datos que pueden ser de distintos tipos. Se puede hacer referencia a toda la estructura como un todo o acceder a sus miembros de forma individual usando un selector de miembro. Además, los structs del mismo tipo pueden ser asignados entre sí, pasados como argumentos o devueltos por funciones, aunque la comparación de structs debe hacerse miembro a miembro.

4. Dar ejemplos de Estructuras o registros

El ejemplo es de Estructura.

Una estructura en C++ es un conjunto de campos (llamados miembros) que pueden ser de diferentes tipos. En el siguiente código, **tPersona** es una estructura con un campo de tipo **string (nombre)** y un campo de tipo **int (edad)**. Se utilizan para agrupar información relacionada de distintos tipos bajo un mismo nombre.

```
#include <iostream>
using namespace std;
// Definimos una estructura llamada 'tPersona'
struct tPersona {
   string nombre;    // Campo para almacenar el nombre (de tipo string)
   int edad;         // Campo para almacenar la edad (de tipo entero)
}
int main() {
 // Declaramos una variable de tipo tPersona llamada 'persona1'
   tPersona persona1;
 // Asignamos valores a los campos de la estructura 'persona1'
   persona1.nombre = "Ana";  // Se almacena el nombre "Ana"
   persona1.edad = 25;       // Se almacena la edad 25

   // Mostramos los valores de la estructura en la consola
   cout << "Nombre: " << persona1.nombre << endl;
   cout << "Edad: " << persona1.edad << endl;
   return 0;
}
```

5. ¿Cómo se pueden Anidar estructuras, ejemplifique?

Las estructuras anidadas permiten definir una estructura dentro de otra, proporcionando una manera eficiente de organizar datos relacionados. En C++, este tipo de estructura facilita la creación de relaciones jerárquicas entre datos.

Ejemplo:

En el siguiente ejemplo vemos como, *la estructura tPersona contiene un campo de tipo tNif, que facilita la organización de la información del NIF dentro de la persona.*

```
#include <iostream>
using namespace std;

//tNif  Estructura que agrupa dos elementos:
struct tNif {
   string dni; //cadena de caracteres de tipo string que almacena el DNI
   char letra;  //representa la letra asociada al NIF
};

struct tPersona {  //estructura que agrupa los datos de una persona
   string nombre; //Cadena que almacena el nombre
   tNif nif; //nif es un miembro de tipo tNif, que almacena la información
};

int main() {
   tPersona persona; //una variable persona de tipo tPersona, q' contiene la info.
//Ejemplo de valores asignando la información de la persona
   persona.nombre = "Carlos";
   persona.nif.dni = "12345678";
   persona.nif.letra = 'A';

//Imprime los datos con la información de la persona
   cout << "Nombre: " << persona.nombre << endl;
   cout << "DNI: " << persona.nif.dni << persona.nif.letra << endl;

  return 0;
}
```

6. ¿Qué es una lista de Longitud variable?

Una lista de longitud variable es una estructura de datos que permite almacenar un número variable de elementos. Esta lista puede crecer o reducirse según sea necesario. Los valores almacenados se asignan a las posiciones del array, y un contador se actualiza para reflejar la cantidad de elementos válidos. Sin embargo, esto puede dar lugar a elementos no utilizados, que ocupan las posiciones restantes del array, generando así valores de datos aleatorios en esas posiciones.

7. ¿Cómo se insertan y eliminan elementos en una lista variable?

Las listas de longitud variable facilitan la inserción y eliminación de elementos sin necesidad de redimensionar todo el array. En estas listas, para insertar un elemento, primero se debe localizar la posición deseada, luego desplazar los elementos posteriores para hacer espacio y finalmente añadir el nuevo elemento, incrementando el contador de elementos válidos. Por otro lado, la eliminación de un elemento disminuye este contador y ajusta las posiciones de los elementos restantes para mantener la coherencia de la lista. La clase std::list en C++ facilita estas operaciones, permitiendo inserciones y eliminaciones eficientes sin necesidad de mover otros elementos.

MARCO PRÁCTICO

Desarrollar un Programa que:

a. Realizar un programa Que rellene un array con los 100 primeros números enteros y los muestre en pantalla
b. Realizar un Programa Que rellene un array con los números primos comprendidos entre 1 y 100 y los muestre en pantalla
c. Realizar un Programa Que rellene un array con los números impares comprendidos entre 1 y 100 y los muestre en pantalla

Lic. Oemig José Luis.