



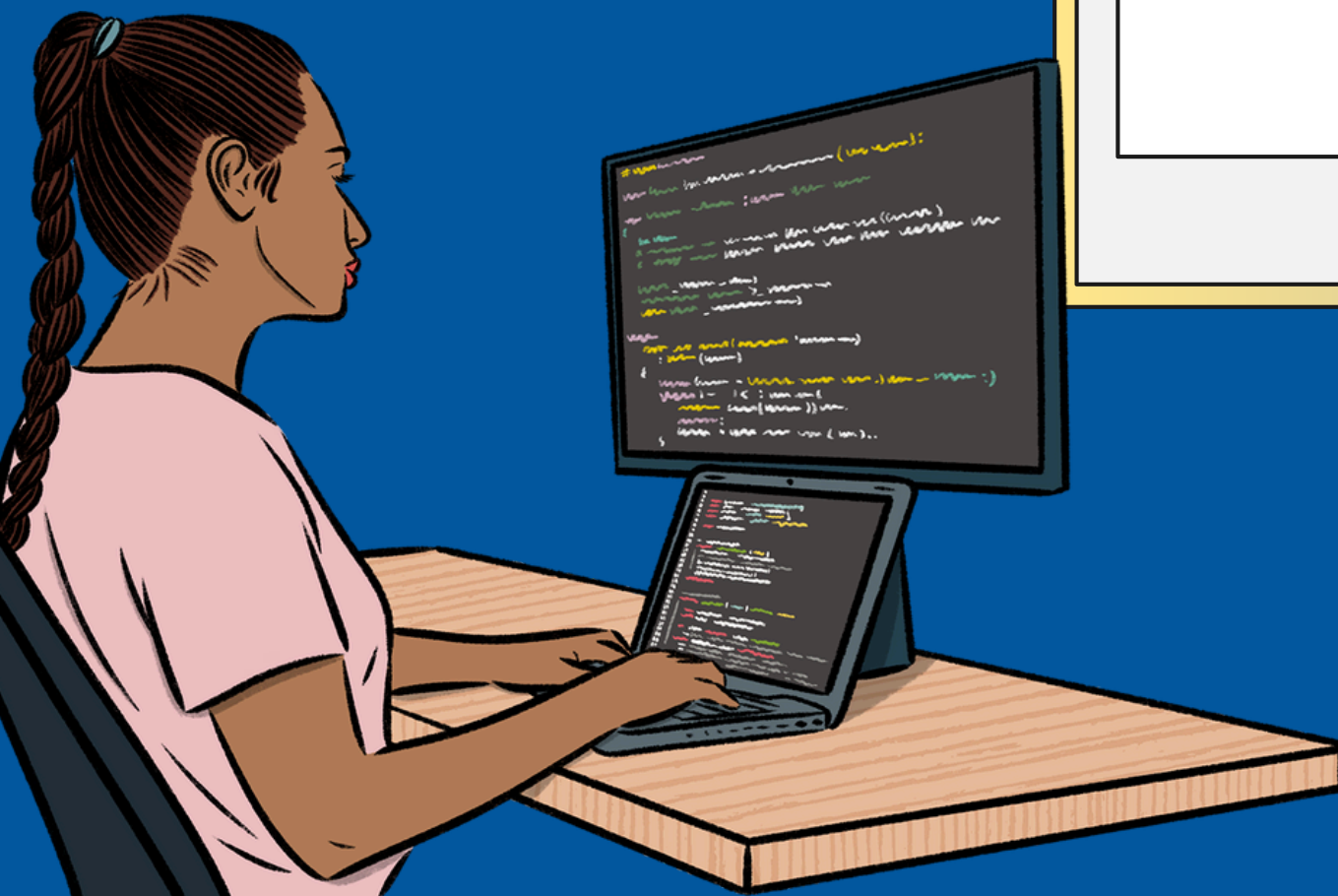
PROGRAMACION AVANZADA

ING. INDUSTRIAL

Esp. Ing.Prof.: Cristina Delia Cruz



TIPOS ESTRUCTURADOS EN C++





TIPOS ESTRUCTURADOS EN C++

Los tipos de datos estructurados en C++ **son aquellos que permiten agrupar múltiples elementos de datos**, que pueden ser del mismo tipo o de tipos diferentes, en una única estructura. Estos tipos de datos son fundamentales cuando se necesita trabajar con grandes conjuntos de datos de forma organizada y eficiente. A continuación, veremos algunos de los principales tipos de datos estructurados en C++.



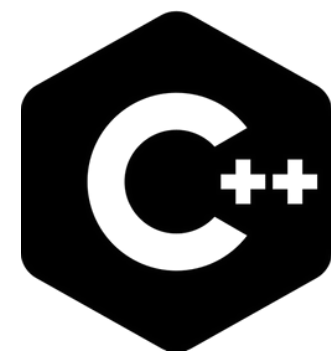
DATOS SIMPLES Y COMPUESTOS

Las variables de tipos de datos simples permiten almacenar un valor específico.



Las variables de tipos de datos compuestos permiten almacenar un grupo de valores.





ARREGLOS EN C++

Un Arreglo es una colección de elementos del mismo tipo almacenados de manera contigua en la memoria. En C++, los arreglos permiten acceder a cada uno de sus elementos mediante índices. Los arreglos son útiles para trabajar con múltiples valores relacionados, pero todos los elementos deben ser del mismo tipo.



ARREGLOS

UNIDIMENSIONALES EN C++

Un arreglo unidimensional (también conocido como vector) **es una colección de elementos del mismo tipo de datos que se almacenan en ubicaciones contiguas de la memoria.** En términos simples, **es una estructura de datos que contiene una lista de valores accesibles a través de un índice.** Los arreglos son ampliamente utilizados en C++ y otros lenguajes de programación para almacenar y gestionar grandes cantidades de datos de manera organizada y eficiente.



ARREGLOS

UNIDIMENSIONALES EN C++

Elementos homogéneos
Todos los elementos dentro del arreglo son del mismo tipo de datos (por ejemplo, todos int, todos float, etc.).



ARREGLOS UNIDIMENSIONALES EN C++

Acceso mediante índices
Cada elemento del arreglo
tiene una posición o índice que
comienza desde 0.



ARREGLOS UNIDIMENSIONALES EN C++

Tamaño fijo

El tamaño del arreglo se define al declararlo y no puede cambiar durante la ejecución del programa (en caso de usar arreglos estáticos).

Sin embargo, en C++, podemos usar la biblioteca estándar (`std::vector`) para tener arreglos dinámicos que pueden redimensionarse.



DECLARACIÓN DE ARREGLOS UNIDIMENSIONALES

En C++, un arreglo unidimensional se declara especificando el tipo de datos y el número de elementos que contendrá.

tipo **nombreArreglo**[**tamaño**];



DECLARACIÓN DE ARREGLOS UNIDIMENSIONALES

tipo nombreArreglo[tamaño];

- **tipo**: Es el tipo de datos que almacenará el arreglo (por ejemplo, **int**, **float**, **char**, etc.).
- **nombreArreglo**: Es el nombre del arreglo.
- **tamaño**: Es el número de elementos que contendrá el arreglo. Debe ser un valor constante o definido durante la declaración.



EJEMPLO DE DECLARACIÓN DE ARREGLO UNIDIMENSIONAL

```
int numeros[5];
```

En este ejemplo, hemos declarado un arreglo llamado numeros que puede almacenar 5 enteros. **Los elementos se acceden mediante sus índices, que van desde 0 hasta 4.**



INICIALIZACIÓN DE ARREGLOS UNIDIMENSIONALES

Un arreglo puede ser inicializado durante su declaración, asignando valores a cada elemento.

```
int numeros[5] = {10, 20, 30, 40, 50};
```

Si no se especifican todos los valores, los elementos restantes se inicializan a 0 de forma predeterminada (en el caso de los enteros).



INICIALIZACIÓN DE ARREGLOS UNIDIMENSIONALES

Para acceder a un elemento de un arreglo, se utiliza el nombre del arreglo seguido del índice entre corchetes “[]”.

```
int numeros[5] = {10, 20, 30, 40, 50};  
int primerElemento = numeros[0];  
int tercerElemento = numeros[2];
```



MODIFICACIÓN DE ELEMENTOS DE UN ARREGLO

Podemos cambiar el valor de un elemento del arreglo usando su índice.

```
numeros[1] = 25;
```

Cambia el segundo elemento de 20 a 25



RECORRER UN ARREGLO

Es común utilizar bucles para recorrer los elementos de un arreglo y realizar operaciones sobre ellos. El bucle más utilizado es for.



RECORRER UN ARREGLO

Ejemplo

```
1  #include <iostream>
2  using namespace std;
3  main() {
4      int numeros[5] = {10, 20, 30, 40, 50};
5
6      for (int i = 0; i < 5; i++) {
7          cout << numeros[i] << " ";
8      }
9      return 0;
10 }
```

En este ejemplo, el bucle recorre todos los elementos del arreglo y los imprime.



LÍMITES DE LOS ARREGLOS

Es importante recordar que los arreglos tienen un tamaño fijo, y acceder a un índice fuera de sus límites puede causar errores o comportamientos inesperados. Este es un error común conocido como desbordamiento de arreglo.



DESVENTAJAS DE LOS ARREGLOS

Tamaño fijo

Una vez que se declara el tamaño del arreglo, no se puede cambiar. Si el tamaño es demasiado grande, se desperdicia memoria, y si es muy pequeño, puede ser insuficiente.

Acceso fuera de límites

C++ no verifica los límites del arreglo en tiempo de ejecución, lo que puede conducir a errores si se intenta acceder a índices no válidos.



CONCLUSION DE LOS ARREGLOS UNIDEMENSIONALES

Los arreglos unidimensionales son una estructura de datos básica y fundamental en C++. Son útiles cuando se requiere almacenar múltiples valores del mismo tipo de manera organizada. Sin embargo, es crucial utilizarlos con cuidado para evitar problemas como el acceso fuera de límites o el mal manejo del tamaño del arreglo. Para más flexibilidad, se puede optar por tipos de datos dinámicos como los vectores.



ARREGLOS BIDIMENSIONALES EN C++

Un arreglo bidimensional en C++ es una estructura de datos que permite almacenar datos en una matriz de filas y columnas. Es una extensión del arreglo unidimensional y puede visualizarse como una **tabla de datos** (como una hoja de cálculo o una tabla de Excel). Es útil cuando se quiere trabajar con relaciones de datos organizados en dos dimensiones, como por ejemplo matrices, tablas de valores o datos tabulares.



DECLARACIÓN DE ARREGLOS BIDIMENSIONALES

Para declarar un arreglo bidimensional en C++, se utiliza la siguiente sintaxis:

```
tipo nombreArreglo[filas][columnas];
```



DECLARACIÓN DE ARREGLOS BIDIMENSIONALES

tipo nombreArreglo[filas][columnas];

- **tipo:** Tipo de dato que almacenará el arreglo (por ejemplo, int, double, char).
- **nombreArreglo:** Nombre del arreglo.
- **filas:** Número de filas del arreglo.
- **columnas:** Número de columnas del arreglo.



EJEMPLO DE DECLARACIÓN

```
int matriz[3][4];
```

Este arreglo contiene 12 elementos (3 filas * 4 columnas). Cada posición en este arreglo puede ser identificada por dos índices: el índice de la fila y el índice de la columna.



ACCESO A LOS ELEMENTOS DEL ARREGLO

Para acceder a un elemento específico del arreglo bidimensional, se utiliza la siguiente notación:

`nombreArreglo[indiceFila][indiceColumna];`

Por ejemplo, si tenemos un arreglo de 3 filas y 4 columnas llamado `matriz` y queremos acceder al valor en la segunda fila y tercera columna, lo haríamos así:

```
int valor = matriz[1][2];
```



INICIALIZACIÓN DE UN ARREGLO BIDIMENSIONAL

Podemos inicializar un arreglo bidimensional de forma manual en el momento de la declaración:

```
int matriz[2][3] = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```



INICIALIZACIÓN DE UN ARREGLO BIDIMENSIONAL

En este caso, tenemos un arreglo con 2 filas y 3 columnas. Los valores se almacenan en el arreglo de la siguiente manera:

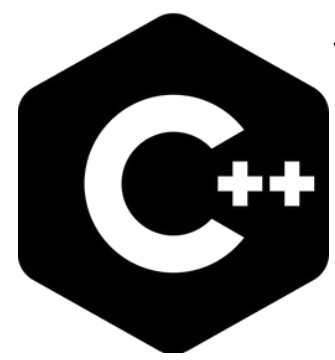
1	2	3
4	5	6



RECORRIDO DE UN ARREGLO BIDIMENSIONAL

Para recorrer todos los elementos de un arreglo bidimensional, se utilizan dos bucles 'for' anidados: uno para recorrer las filas y otro para las columnas.

```
for (int i = 0; i < filas; i++) {  
    for (int j = 0; j < columnas; j++) {  
        cout << matriz[i][j] << " ";  
    }  
    cout << endl;  
}
```



VENTAJAS DE LOS ARREGLOS BIDIMENSIONALES

1. **Organización de datos:** Permiten organizar datos en una estructura tabular, lo que facilita su manejo y comprensión.
2. **Acceso directo:** Permiten el acceso rápido a cualquier elemento usando índices de filas y columnas.
3. **Facilidad para realizar operaciones matemáticas:** En campos como la Ingeniería Industrial, las matrices son esenciales para cálculos de optimización, costos, y operaciones en general.



LIMITACIONES

1. **Tamaño fijo:** Una vez que se declara un arreglo bidimensional, su tamaño es fijo y no se puede cambiar durante la ejecución del programa.
2. **Uso de memoria:** Los arreglos bidimensionales ocupan más memoria, lo que puede ser un problema en programas que manejan grandes cantidades de datos.



CONCLUSION

Los arreglos bidimensionales son muy versátiles y se utilizan ampliamente en problemas donde se necesita almacenar y manipular datos en dos dimensiones.



ARREGLOS DE CARACTERES

En C++, un arreglo de caracteres es una secuencia de caracteres almacenados en posiciones consecutivas en memoria. Los arreglos de caracteres se usan comúnmente para representar cadenas de texto. Aquí están algunas operaciones comunes:



ARREGLOS DE CARACTERES

Puedes definir e inicializar un arreglo de caracteres de esta forma:

```
char arreglo[] = "Hola Mundo"; // El tamaño del arreglo se ajusta automáticamente
```

O especificar el tamaño:

```
char arreglo[20] = "Hola Mundo"; // El tamaño es 20, suficiente para "Hola Mundo" y el carácter nulo
```

Acceso a Elementos: Puedes acceder a elementos individuales del arreglo usando el índice:

```
char arreglo[] = "Hola";
```

```
char primerCaracter = arreglo[0]; // 'H'
```

Modificación de Elementos: Puedes modificar caracteres específicos del arreglo:

```
char arreglo[] = "Hola";
```

```
arreglo[0] = 'h'; // Cambia 'H' a 'h'
```



ARREGLOS DE CARACTERES

Longitud de una Cadena: La longitud de un arreglo de caracteres se puede encontrar usando la función `strlen()` de `<cstring>`:

```
#include <iostream>
#include <cstring>

int main() {
    char arreglo[] = "Hola Mundo";
    int longitud = strlen(arreglo); // 10
    cout << "Longitud: " << longitud << endl;
    return 0;
}
```




ARREGLOS DE CARACTERES

Comparación de Cadenas: Para comparar dos arreglos de caracteres (cadenas), puedes usar `strcmp()` de `<cstring>`:

```
#include <iostream>
#include <cstring>

int main() {
    char cadena1[] = "Hola";
    char cadena2[] = "Hola";
    if (strcmp(cadena1, cadena2) == 0) {
        cout << "Las cadenas son iguales." << endl;
    } else {
        cout << "Las cadenas son diferentes." << endl;
    }
    return 0;
}
```




ARREGLOS DE CARACTERES

Copia de Cadenas: Para copiar el contenido de una cadena a otra, usa strcpy():

```
#include <iostream>
#include <cstring>

int main() {
    char origen[] = "Hola Mundo";
    char destino[20];
    strcpy(destino, origen); // Copia "Hola Mundo" a destino
    cout << "Destino: " << destino << endl;
    return 0;
}
```



ARREGLOS CON CADENAS

En C++, también puedes tener arreglos de cadenas (arreglos de arreglos de caracteres). Esto es útil para almacenar varias cadenas de texto.

Definición e Inicialización: Puedes definir un arreglo de cadenas de esta manera:

```
char cadenas[3][20] = { "Primera", "Segunda", "Tercera" };
```



ARREGLOS CON CADENAS

Acceso a Elementos: Puedes acceder a cadenas específicas e incluso a caracteres dentro de esas cadenas:

```
#include <iostream>

int main() {
    char cadenas[3][20] = { "Primera", "Segunda", "Tercera" };
    cout << "Primera cadena: " << cadenas[0] << endl;
    cout << "Caracter en la segunda cadena: " << cadenas[1][2] << endl;
    return 0;
}
```



ARREGLOS CON CADENAS

Modificación de Elementos: Puedes modificar caracteres específicos en un arreglo de cadenas:

```
#include <iostream>

int main() {
    char cadenas[3][20] = { "Primera", "Segunda", "Tercera" };
    cadenas[0][0] = 'p'; // Cambia 'P' a 'p' en la primera cadena
    cout << "Primera cadena modificada: " << cadenas[0] << endl;
    return 0;
}
```



ARREGLOS CON CADENAS

Lectura de cadenas

El objeto **cin** del flujo de entrada incluye entre otras las funciones de entrada:

cin.getline(): permite a cin leer la cadena completa, incluyendo cualquier espacio en blanco

cin.get(): se utiliza para leer carácter a carácter

cin.ignore(): lee uno o más caracteres del flujo de entrada cin sin procesar

Nota: también puede utilizarse **fflush(stdin)** para limpiar el **buffer de entrada**



ARREGLOS CON CADENAS

Para trabajar con estas funciones, al comienzo del programa hay que escribir

```
#include <string.h>
```

La biblioteca `<string.h>` contiene un conjunto de funciones para manipular cadenas: copiar, cambiar caracteres, comparar cadenas, etc.

Las funciones más elementales son:

strcpy (c1, c2); Copia c2 en c1

strcat (c1, c2); Añade c2 al final de c1

strlen (cadena); Devuelve la longitud de la cadena

strcmp (c1, c2); Devuelve cero si c1 es igual a c2; no-cero en caso contrario

strlwr(c1); Convierte todos los caracteres de la cadena c1 a minúscula

strupr(c1); Convierte todos los caracteres de la cadena c1 a mayúsculas



REGISTROS EN C++

Un registro en C++ es una estructura de datos que agrupa varios campos (variables) de diferentes tipos bajo una sola entidad, permitiendo acceder y manipularlos de manera organizada. Por ejemplo, puedes tener un registro llamado Empleado que contiene campos como el nombre, el salario, y el ID de empleado.



REGISTROS EN C++

Definición de una Estructura

Para definir un registro en C++, utilizamos la palabra clave **struct** seguida del “**nombre de la estructura**”. Dentro de las llaves {}, declaramos los miembros (variables) que componen la estructura.



REGISTROS EN C++

Registros/Estructuras

Los registros, también llamados estructuras (**struct**) en C++, son una forma de agrupar varios datos relacionados bajo un mismo nombre.

A diferencia de los arreglos, que solo pueden contener elementos del mismo tipo, una estructura puede agrupar diferentes tipos de datos (por ejemplo, un número entero, un carácter y un número con decimales) en una sola unidad. Cada uno de estos datos individuales dentro de la estructura se conoce como miembro o campo. Piensa en una estructura como una ficha o un formulario que contiene información completa sobre una sola entidad, como un empleado, un producto o un cliente. Por ejemplo, en lugar de tener variables separadas para el nombre, el apellido y la edad de una persona, puedes agruparlas todas en una única estructura llamada Persona.



REGISTROS EN C++

Acceso a los Miembros de la Estructura

Como se ve en el ejemplo 1, los miembros de una estructura se acceden usando el operador de punto (.). Es decir, si tienes una variable de tipo Empleado, puedes acceder a sus miembros usando `nombre_variable.miembro`.



REGISTROS EN C++

Estructuras y Funciones

También puedes pasar estructuras a funciones para trabajar con los datos. Pueden pasarse por valor o por referencia, dependiendo de si deseas modificar el registro original o solo trabajar con una copia.



REGISTROS EN C++

Uso de Arreglos de Estructuras

En C++, también puedes tener arreglos de estructuras, lo que te permite manejar varios registros del mismo tipo.



REGISTROS EN C++

Características de las Estructuras en C++:

1. Tipos heterogéneos: Puedes almacenar diferentes tipos de datos bajo el mismo registro (entero, cadena, flotante, etc.).
2. Memoria contigua: Los miembros de una estructura se almacenan en posiciones de memoria contiguas.
3. Pasar a funciones: Las estructuras pueden pasarse por valor o por referencia a funciones.
4. Acceso mediante el operador punto: Se accede a los miembros utilizando el operador ".".



REGISTROS EN C++

*En C++, los registros se implementan mediante estructuras (**struct**), lo que permite agrupar diferentes tipos de datos bajo una misma entidad lógica. Son útiles cuando quieres manejar datos relacionados, como la información de una persona, un producto, o un empleado, de manera estructurada y más organizada.*



REGISTROS EN C++

Tamaño de una estructura

El tamaño de una estructura es la cantidad de memoria que el programa necesita para guardar todos sus miembros.

*Imagina una estructura llamada **Paquete** para un almacén. Esta estructura tiene dos miembros:*

- ***Un código de producto (int)**, que ocupa 4 bytes.*
- ***El peso del paquete (double)**, que ocupa 8 bytes.*

*Aunque la suma lógica es 12 bytes, al usar **sizeof(Paquete)**, el resultado podría ser **16 bytes**. Esto se debe a un concepto llamado "**relleno**" (**padding**), donde el compilador alinea los datos en la memoria para que el procesador pueda leerlos más rápido y eficientemente. En este caso, el compilador podría **añadir 4 bytes** de espacio extra después del int para que el double se inicie en una dirección de memoria que sea un múltiplo de 8, lo cual mejora el rendimiento de la CPU.*

*En resumen, **el tamaño de la estructura es la suma de los tamaños de todos sus campos, más el espacio adicional que el compilador agrega por cuestiones de optimización.***



EJEMPLOS

Ejemplo 1

Análisis de Producción Diaria

Desarrolla un programa modular para analizar la producción de una semana. Crea una función llamada `cargarProduccionSemanal` que reciba un arreglo de 7 enteros por referencia y lo llene con los datos de producción de cada día. Luego, crea una segunda función llamada `calcularPromedio` que reciba el arreglo y devuelva la producción total y el promedio diario. El programa principal solo debe llamar a estas funciones para ejecutar el proceso.



EJEMPLOS

Ejemplo 2

Estatus de Ensamble de Componentes

Diseña un programa modular para registrar y analizar el estatus de 5 componentes. Crea una función `cargarEstatus` que reciba un arreglo de caracteres por referencia para llenarlo con los estados de los componentes. Luego, crea una función `contarAprobados` que reciba el arreglo y devuelva el número total de componentes con estatus 'A' o 'a'.

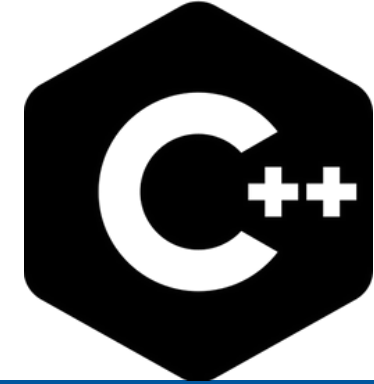


EJEMPLOS

Ejemplo 3

Matriz de Producción por Turno y Línea

Implementa un programa modular para gestionar una matriz de producción de 3x4. Crea una función `cargarMatrizProduccion` que reciba la matriz por referencia y la llene con datos ingresados por el usuario. Luego, crea una función `mostrarMatriz` para imprimir la tabla completa. Finalmente, crea una función `calcularTotalLinea` que reciba la matriz, el número de la línea y devuelva el total de unidades producidas en esa línea.

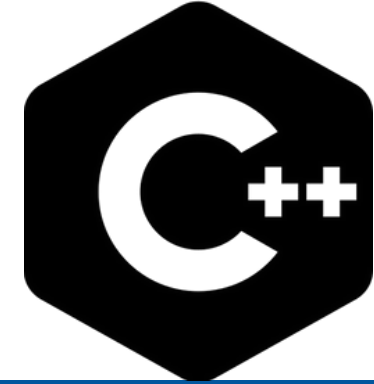


EJEMPLOS

Ejemplo 4

Tabla de Inspección de Calidad

Modulariza un programa para generar una tabla de inspección de calidad. Crea una función `inicializarTabla` que reciba una matriz de caracteres por referencia y la llene con los datos de inspección predefinidos. Luego, crea una función `imprimirInforme` que reciba la matriz y la muestre en un formato legible, convirtiendo los caracteres 'O' y 'F' en "OK" y "FAIL" respectivamente.



EJEMPLOS

Ejemplo 5

Gestión de Información de Productos

Define una estructura `Producto` con sus campos. Crea una función `ingresarProducto` que reciba una variable `Producto` por referencia y pida al usuario que llene sus campos. Luego, crea una función `mostrarProducto` que reciba una variable `Producto` y muestre toda su información de forma ordenada. El programa principal solo debe contener las llamadas a estas dos funciones.



EJEMPLOS

Ejemplo 6

Registro de Empleado y Horario de Turno

Define dos estructuras: Horario y Empleado, con Horario anidada en Empleado. Crea una función ingresarEmpleado que reciba una variable Empleado por referencia y solicite los datos del empleado y su horario. Luego, crea una función mostrarFichaEmpleado para imprimir toda la información, demostrando el acceso a los miembros de la estructura anidada.



EJEMPLOS

Ejemplo 7

Control de Stock de un Almacén (Arreglo de Registros)

Utiliza la estructura Producto del ejemplo anterior. Crea una función cargarAlmacen que reciba un arreglo de 3 productos por referencia y lo llene con datos. Luego, crea una función buscarProducto que reciba el arreglo y un código de producto, y devuelva la posición en la que se encuentra. Finalmente, una función mostrarInventario para imprimir la información de todos los productos en el arreglo.



EJEMPLOS

Ejemplo 8

Registro de Máquina con Mantenimientos Anuales

Define una estructura Maquina que incluya un arreglo para los 12 mantenimientos mensuales. Crea una función ingresarMantenimientos que reciba una variable Maquina por referencia y pida los datos de la máquina, incluyendo la carga de mantenimientos para cada mes. Luego, crea una función calcularTotalAnual que reciba el arreglo de mantenimientos y devuelva la suma total. Finalmente, el programa principal debe usar estas funciones para mostrar un resumen anual.

¿PREGUNTAS?

```
render() {  
  return (  
    <React.Fragment>  
      <div className="py-5">  
        <div className="container">  
          <Title name="our" title="product">  
            <div className="row">  
              <ProductConsumer>  
                {(value) => {  
                  console.log(value)  
                }}  
            </ProductConsumer>  
          </div>  
        </div>  
      </div>  
    </React.Fragment>  
  )  
}
```




MUCHAS
GRACIAS!!!

Nos Vemos en la Siguiente Clase