Unidad 4: Organización de Datos

4.1 Arreglos

Un **arreglo** es una estructura que nos permite almacenar una colección de elementos del mismo tipo en una sola variable. La ventaja de un arreglo es que podemos gestionar y acceder a grandes cantidades de datos de forma organizada. Pensemos en el arreglo como una lista de casillas numeradas, donde cada casilla contiene un dato específico y se puede acceder a él por su número de posición, llamado *índice*.

Ejemplo básico en DevC++:

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

int main() {
  int numeros[5] = {10, 20, 30, 40, 50};
  cout << "El primer elemento es: " << numeros[0] << endl; // Acceso al primer elemento
  return 0;
}
```

En este ejemplo, hemos definido un arreglo de 5 elementos. Los índices de los arreglos empiezan en 0, por lo tanto, numeros[0] representa el primer elemento (10).

4.2 Arreglos Unidimensionales: Conceptos Básicos, Operaciones y Aplicaciones

Un **arreglo unidimensional** es como una lista o fila de datos. Pueden realizar varias operaciones básicas, como agregar, modificar, y acceder a valores individuales.

Ejercicio

Imaginemos que tenemos que almacenar y sumar las edades de 5 personas:

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

int main() {
  int edades[5] = {18, 22, 25, 20, 19}; // Declaración y asignación de valores
  int suma = 0;

  for(int i = 0; i < 5; i++) { // Ciclo para recorrer el arreglo
    suma += edades[i];
  }

  cout << "La suma de las edades es: " << suma << endl;
```

```
  return 0;
}
```

Este ejemplo usa un **ciclo for** para recorrer el arreglo y sumar cada elemento.

Aplicaciones

- Almacenar datos en serie, como las temperaturas de una semana.
- Sumar valores o realizar cálculos sobre una lista de datos.

4.3 Arreglos Multidimensionales: Conceptos Básicos, Operaciones y Aplicaciones

Los **arreglos multidimensionales** (matrices) son extensiones de los arreglos unidimensionales, donde cada elemento puede tener múltiples índices. Un **arreglo bidimensional**, por ejemplo, puede pensarse como una tabla con filas y columnas.

Ejemplo en DevC++

Supongamos que necesitamos almacenar las notas de 3 estudiantes en 4 asignaturas:

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

int main() {
  int notas[3][4] = {
    {85, 90, 88, 92},
    {78, 81, 86, 79},
    {90, 92, 93, 88}
  };

  for(int i = 0; i < 3; i++) { // Filas
    for(int j = 0; j < 4; j++) { // Columnas
      cout << "Estudiante " << i+1 << ", Asignatura " << j+1 << ": " << notas[i][j] << endl;
    }
  }
  return 0;
}
```

Aquí vemos un arreglo bidimensional de 3x4. Noten cómo accedemos a cada elemento con dos índices: uno para la fila y otro para la columna.

Aplicaciones

- Almacenar y procesar datos tabulares (como una hoja de cálculo).
- Modelar tableros de juegos, horarios o tablas de valores.

Arreglos char

Los **arreglos de tipo char** son muy útiles para almacenar cadenas de caracteres, como palabras o frases. Los caracteres (char) se manejan de forma similar a otros tipos de datos, pero cada elemento del arreglo representa un carácter individual. En C++, también se usan los arreglos de char para manejar cadenas de texto de forma simple.

Ejemplo 1: Arreglo de caracteres para almacenar una palabra

Vamos a crear un arreglo de char que almacene la palabra "Hola".

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

int main() {
  char saludo[5] = {'H', 'o', 'l', 'a', '\0'}; // '\0' es el carácter nulo que indica el final de la cadena

  cout << "La palabra es: " << saludo << endl;

  return 0;
}
```

Explicación

- Aquí, saludo es un arreglo de 5 elementos de tipo char.
- El último elemento '\0' es el **carácter nulo**, el cual indica el fin de la cadena y permite que cout sepa dónde detenerse al imprimir.
- Al ejecutar el programa, se imprimirá la palabra "Hola".

Ejemplo 2: Lectura de una palabra ingresada por el usuario

Veamos cómo leer una palabra ingresada por el usuario y almacenarla en un arreglo de char.

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

int main() {
  char nombre[20];

  cout << "Ingresa tu nombre: ";
  cin >> nombre;

  cout << "Hola, " << nombre << "!" << endl;

  return 0;
```

```
}
```

Explicación

- Aquí, nombre es un arreglo de char con capacidad para 20 caracteres.
- cin lee la entrada del usuario y la almacena en el arreglo nombre. Noten que cin detiene la lectura al encontrar un espacio, por lo que solo captura la primera palabra si el usuario escribe varias.

Ejemplo 3: Arreglo de char multidimensional para almacenar varias palabras

Ahora, crearemos un arreglo de char bidimensional para almacenar múltiples palabras, como nombres de estudiantes.

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

int main() {
  char nombres[3][10] = {"Ana", "Luis", "Pedro"}; // Cada palabra puede tener hasta 9 letras + el '\0'

  cout << "Lista de nombres:" << endl;
  for(int i = 0; i < 3; i++) {
    cout << nombres[i] << endl;
  }

  return 0;
}
```

Explicación

- nombres es un arreglo bidimensional de char que almacena 3 nombres, cada uno de hasta 10 caracteres (incluido '\0').
- El ciclo for recorre cada fila del arreglo nombres e imprime cada nombre.

Ejemplo 4: Concatenar dos cadenas almacenadas en arreglos de char

Concatenaremos dos palabras ingresadas por el usuario en un solo arreglo de char. Usaremos la función strcat de la biblioteca <cstring> para esto.

```
#include <iostream>
#include <cstring> // Para usar strcat
using namespace std;

int main() {
  char palabra1[20], palabra2[20];
```

```
  cout << "Ingresa la primera palabra: ";
  cin >> palabra1;
  cout << "Ingresa la segunda palabra: ";
  cin >> palabra2;

  strcat(palabra1, palabra2); // Concatenamos palabra2 al final de palabra1

  cout << "Palabras concatenadas: " << palabra1 << endl;

  return 0;
}
```

Explicación

- palabra1 y palabra2 son arreglos de char para almacenar las palabras ingresadas.
- strcat(palabra1, palabra2) agrega el contenido de palabra2 al final de palabra1, y el resultado se almacena en palabra1.
- cout luego imprime la cadena concatenada.

Estos ejemplos muestran cómo los arreglos de char son útiles para almacenar y manipular texto. Prueben modificarlos y experimenten con diferentes palabras para consolidar su comprensión. ¡A seguir practicando!

Ejemplo de concatenación concatenando los dos primeros arreglos en un tercero y por medio de un ciclo en lugar de strcat. La idea es copiar cada carácter del primer arreglo al tercer arreglo y luego hacer lo mismo con el segundo arreglo.
Aquí está el ejemplo modificado:

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

int main() {
  char palabra1[20], palabra2[20], palabraConcatenada[40]; // Tercer arreglo para almacenar la concatenación

  cout << "Ingresa la primera palabra: ";
  cin >> palabra1;
  cout << "Ingresa la segunda palabra: ";
  cin >> palabra2;

  // Copiar palabra1 a palabraConcatenada
```

```
  int i = 0;
  while (palabra1[i] != '\0') { // Copia cada carácter hasta el final de palabra1
    palabraConcatenada[i] = palabra1[i];
    i++;
  }

  // Concatenar palabra2 en palabraConcatenada
  int j = 0;
  while (palabra2[j] != '\0') { // Copia cada carácter de palabra2 al final de palabraConcatenada
    palabraConcatenada[i] = palabra2[j];
    i++;
    j++;
  }

  // Agregar el carácter nulo al final de palabraConcatenada
  palabraConcatenada[i] = '\0';

  cout << "Palabras concatenadas: " << palabraConcatenada << endl;

  return 0;
}
```

Explicación

1. **Copiar palabra1 a palabraConcatenada**:
    - Usamos un ciclo while que recorre palabra1 y copia cada carácter en palabraConcatenada hasta que encuentra el carácter nulo ('\0'), que indica el final de palabra1.
    - La variable i se usa tanto para movernos en palabra1 como para indicar la posición actual en palabraConcatenada.
2. **Concatenar palabra2 en palabraConcatenada**:
    - Después de copiar palabra1, i ya apunta al primer índice vacío de palabraConcatenada.
    - Con otro ciclo while, copiamos cada carácter de palabra2 en palabraConcatenada, incrementando i y j en cada iteración hasta alcanzar el carácter nulo de palabra2.
3. **Agregar el carácter nulo '\0'**:
    - Al final de palabraConcatenada, colocamos el carácter '\0' para indicar el final de la cadena.

Ejemplo de salida

Ingresa la primera palabra: Hola
Ingresa la segunda palabra: Mundo
Palabras concatenadas: HolaMundo

4.4 Estructuras o Registros

Las **estructuras** permiten almacenar varios datos de tipos distintos bajo un mismo nombre. Imaginen una estructura como una pequeña base de datos dentro de un programa, donde cada elemento contiene múltiples atributos.

Ejemplo de Estructura en DevC++

Vamos a definir una estructura para almacenar datos de estudiantes:

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

struct Estudiante {
  string nombre;
  int edad;
  float promedio;
};

int main() {
  Estudiante estudiante1;
  estudiante1.nombre = "Juan";
  estudiante1.edad = 20;
  estudiante1.promedio = 8.5;

  cout << "Nombre: " << estudiante1.nombre << endl;
  cout << "Edad: " << estudiante1.edad << endl;
  cout << "Promedio: " << estudiante1.promedio << endl;

  return 0;
}
```

En este ejemplo, creamos una estructura llamada Estudiante con tres atributos: nombre, edad, y promedio. Luego, definimos un estudiante1 que tiene un nombre, edad y promedio.

Aplicaciones

- Representar información compleja en un solo objeto, como datos de estudiantes, productos, o libros.
- Organizar datos de distintos tipos en un solo lugar, lo cual hace el código más estructurado y fácil de entender.

Consejos Prácticos

1. **Practiquen ejemplos pequeños** para familiarizarse con la sintaxis.
2. **Usen comentarios** en el código para recordar el propósito de cada sección.
3. **Hagan pruebas** ingresando valores en los arreglos y estructuras para ver cómo responden.
4. **Recuerden formatear su código** correctamente y dividir el problema en pasos más pequeños, especialmente en ejercicios con arreglos o estructuras más complejos.

Arreglos dentro de una estructura

Es posible tener un arreglo dentro de una estructura en C++. De hecho, es una técnica común cuando necesitas que una estructura almacene múltiples valores del mismo tipo.

Veamos un ejemplo sencillo para ilustrar cómo un arreglo puede estar dentro de una estructura.

Ejemplo: Registro de un Estudiante con Calificaciones

Supongamos que queremos definir una estructura para almacenar los datos de un estudiante, incluyendo un arreglo para almacenar sus calificaciones en diferentes materias.

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

// Definición de la estructura Estudiante
struct Estudiante {
  string nombre;
  int edad;
  int calificaciones[5];  // Arreglo para almacenar las calificaciones de 5 materias
};

int main() {
  Estudiante estudiante1; // Declaramos una variable de tipo Estudiante
```

```
  // Ingreso de datos
  cout << "Ingresa el nombre del estudiante: ";
  getline(cin, estudiante1.nombre);
  cout << "Ingresa la edad del estudiante: ";
  cin >> estudiante1.edad;

  // Ingresamos las calificaciones
  cout << "Ingresa las calificaciones de 5 materias:\n";
  for (int i = 0; i < 5; i++) {
    cout << "Calificación " << i + 1 << ": ";
    cin >> estudiante1.calificaciones[i];
  }

  // Mostramos la información ingresada
  cout << "\nInformación del estudiante:\n";
  cout << "Nombre: " << estudiante1.nombre << endl;
  cout << "Edad: " << estudiante1.edad << endl;
  cout << "Calificaciones: ";
  for (int i = 0; i < 5; i++) {
    cout << estudiante1.calificaciones[i] << " ";
  }
  cout << endl;

  return 0;
}
```

Explicación

1. **Definición de la Estructura**:
   - La estructura Estudiante contiene un string para el nombre, un int para la edad y un arreglo int calificaciones[5] para las calificaciones de 5 materias.
2. **Uso de la Estructura**:
   - Creamos una variable estudiante1 de tipo Estudiante para almacenar los datos de un estudiante.
   - El programa solicita el nombre, la edad y las 5 calificaciones.
3. **Acceso a los Datos**:
   - Usamos un bucle for para llenar y luego mostrar los elementos del arreglo calificaciones dentro de la estructura.

Ejemplo de Salida

Ingresa el nombre del estudiante: Carlos Lopez
Ingresa la edad del estudiante: 20
Ingresa las calificaciones de 5 materias:
Calificación 1: 85
Calificación 2: 90
Calificación 3: 78
Calificación 4: 88
Calificación 5: 92

Información del estudiante:
Nombre: Carlos Lopez
Edad: 20
Calificaciones: 85 90 78 88 92

Ventajas de Arreglos en Estructuras

- **Organización**: Ayuda a organizar la información relacionada, como almacenar todas las calificaciones de un estudiante en un solo lugar.
- **Fácil Acceso**: Permite el acceso a cada valor del arreglo usando un índice, como en estudiante1.calificaciones[i].
- **Extensible**: Puede agregar otros arreglos o variables en la estructura para manejar más información de manera ordenada.

Esta técnica es útil para modelar datos en programas más complejos, como bases de datos de estudiantes, inventarios, entre otros.

Arreglos de estructuras

También es posible tener un arreglo de estructuras en C++. Esta técnica es muy útil cuando necesitas manejar un conjunto de datos relacionados, por ejemplo, varios estudiantes, productos en un inventario, empleados en una empresa, etc.

Ejemplo: Arreglo de Estudiantes

Supongamos que queremos almacenar información de varios estudiantes, como sus nombres, edades y calificaciones en diferentes materias. Podemos crear un arreglo de estructuras para gestionar la información de cada estudiante.

Aquí tienes un ejemplo:

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

// Definición de la estructura Estudiante
struct Estudiante {
```

```
    string nombre;
    int edad;
    int calificaciones[3];  // Arreglo para las calificaciones de 3 materias
};

int main() {
    const int NUM_ESTUDIANTES = 3; // Número de estudiantes a almacenar
    Estudiante estudiantes[NUM_ESTUDIANTES]; // Arreglo de estructuras Estudiante

    // Ingreso de datos para cada estudiante
    for (int i = 0; i < NUM_ESTUDIANTES; i++) {
        cout << "\nIngresa los datos del estudiante " << i + 1 << ":\n";

        cout << "Nombre: ";
        cin.ignore();  // Ignorar salto de línea pendiente
        getline(cin, estudiantes[i].nombre);

        cout << "Edad: ";
        cin >> estudiantes[i].edad;

        // Ingresamos las calificaciones
        cout << "Ingresa las calificaciones de 3 materias:\n";
        for (int j = 0; j < 3; j++) {
            cout << "Calificación " << j + 1 << ": ";
            cin >> estudiantes[i].calificaciones[j];
        }
    }

    // Mostramos la información de todos los estudiantes
    cout << "\nInformación de los estudiantes:\n";
    for (int i = 0; i < NUM_ESTUDIANTES; i++) {
        cout << "\nEstudiante " << i + 1 << ":\n";
        cout << "Nombre: " << estudiantes[i].nombre << endl;
        cout << "Edad: " << estudiantes[i].edad << endl;

        cout << "Calificaciones: ";
        for (int j = 0; j < 3; j++) {
            cout << estudiantes[i].calificaciones[j] << " ";
```

```
    }
    cout << endl;
  }

  return 0;
}
```

Explicación del Código

1. **Definición de la Estructura**:
    - La estructura Estudiante contiene string nombre, int edad y un arreglo int calificaciones[3] para almacenar 3 calificaciones.
2. **Arreglo de Estructuras**:
    - Declaramos un arreglo estudiantes de tipo Estudiante con NUM_ESTUDIANTES elementos.
    - Cada elemento del arreglo estudiantes[i] es una estructura de tipo Estudiante.
3. **Ingreso de Datos**:
    - En un bucle for, pedimos al usuario que ingrese los datos de cada estudiante, incluyendo nombre, edad y sus 3 calificaciones.
4. **Mostrar Datos**:
    - Otro bucle for recorre el arreglo estudiantes para mostrar la información almacenada de cada estudiante.

Ejemplo de Salida

Ingresa los datos del estudiante 1:
Nombre: Ana Martínez
Edad: 19
Ingresa las calificaciones de 3 materias:
Calificación 1: 85
Calificación 2: 90
Calificación 3: 88

Ingresa los datos del estudiante 2:
Nombre: Luis García
Edad: 20
Ingresa las calificaciones de 3 materias:
Calificación 1: 78
Calificación 2: 85
Calificación 3: 80

Ingresa los datos del estudiante 3:
Nombre: Carla Reyes
Edad: 21
Ingresa las calificaciones de 3 materias:
Calificación 1: 92
Calificación 2: 88
Calificación 3: 91

Información de los estudiantes:

Estudiante 1:
Nombre: Ana Martínez
Edad: 19
Calificaciones: 85 90 88

Estudiante 2:
Nombre: Luis García
Edad: 20
Calificaciones: 78 85 80

Estudiante 3:
Nombre: Carla Reyes
Edad: 21
Calificaciones: 92 88 91

Ventajas del Arreglo de Estructuras

- **Organización**: Cada elemento en el arreglo representa un registro completo de un estudiante, lo que organiza los datos de manera clara y accesible.
- **Fácil Acceso**: Puedes acceder a cualquier estudiante usando su índice (estudiantes[i]) y, dentro de ese estudiante, acceder a sus propiedades (nombre, edad, calificaciones).
- **Escalabilidad**: Si el número de estudiantes aumenta, solo necesitas ajustar el tamaño del arreglo, y la estructura del programa puede permanecer igual.

Este patrón es útil para manejar grandes cantidades de datos en programas de administración de información.