Colas en C++

Para trabajar con colas en C++, es importante tener claros ciertos temas previos que servirán como base para entender e implementar esta estructura de datos. A continuación, te menciono los temas fundamentales:

1. Fundamentos de Programación en C++
Variables y Tipos de Datos: Comprender cómo declarar variables, los diferentes tipos de datos en C++ (int, float, char, etc.), y cómo se manipulan.
Operadores: Saber usar operadores aritméticos (+, -, *, /), de comparación (==, !=, >, <), y lógicos (&&, ||, !).

2. Estructuras de Control
Condicionales: Usar sentencias como if, else if, else, para la toma de decisiones.
Bucles: Entender cómo funcionan los bucles for, while, y do-while para iterar sobre elementos de la cola.
switch: Si *bien no es fundamental*, puede ser útil en ciertos contextos para manejar diferentes casos.

3. Funciones en C++
Declaración y Definición de Funciones: Conocer cómo crear y usar funciones para modularizar el código.
Paso por Valor y Paso por Referencia: Saber cuándo pasar una variable por valor o por referencia a una función.
Recursión: Aunque las colas no suelen usar recursión de forma directa, es importante comprender este concepto para otras estructuras como árboles o algoritmos relacionados.

4. Arreglos (Arrays)
Declaración y Uso de Arrays: Las colas, especialmente en su implementación manual (como la que vimos), a menudo utilizan arrays para almacenar los elementos.
Operaciones con Arrays: Saber cómo recorrer un array, acceder a elementos específicos, y modificar valores.

5. Estructuras (struct)
Declaración de struct: Comprender cómo crear y utilizar estructuras en C++. Esto es fundamental para implementar una cola con struct, como vimos en el ejemplo.
Miembros de struct: Entender cómo acceder y modificar los miembros de una estructura.

6. Punteros y Referencias
Punteros Básicos: Aunque en una cola simple con arrays no es imprescindible, si trabajas con colas dinámicas (como las implementadas con listas enlazadas), necesitarás saber cómo usar punteros.
Punteros a struct: Si implementas una cola dinámica, necesitarás manipular punteros para enlazar nodos.
Operador ->: Conocer cómo acceder a los miembros de un struct a través de un puntero usando el operador flecha (->).

7. Estructuras Dinámicas (Opcional)
Memoria Dinámica (new y delete): Si planeas implementar una cola dinámica (una cola que crezca o se reduzca en tiempo de ejecución), es importante conocer cómo gestionar memoria de manera dinámica en C++.
Listas Enlazadas: Las colas pueden implementarse de forma dinámica usando listas enlazadas, por lo que es útil conocer cómo funcionan estas estructuras.

8. Biblioteca Estándar STL
std::queue: Saber utilizar la implementación de colas de la biblioteca estándar de C++ te facilitará el trabajo sin necesidad de reinventar el código manualmente.
Métodos de STL (Como push, pop, front, back, etc.): Es fundamental entender las operaciones básicas que ofrece la clase queue de la STL para realizar inserciones, eliminaciones y obtener el elemento del frente.

9. Algoritmos Básicos
Orden y Complejidad: Tener un conocimiento básico sobre la eficiencia de las operaciones, aunque no es esencial para comenzar, te ayudará a entender la eficiencia de las colas.

Conclusión:

Estos son los *temas clave que deberías tener claros* antes de trabajar con colas en C++. Al comprender estos conceptos, estarás preparado para manejar tanto la implementación manual como la que ofrece la biblioteca estándar de C++.

Te proporcionaré una explicación detallada sobre las colas en C++, incluyendo cómo crearlas, insertar elementos, eliminar elementos y cómo operan en general.

¿Qué es una Cola?

Se puede implementar y trabajar con colas en C++ **sin utilizar la biblioteca STL (Standard Template Library)**. Al hacerlo, se pueden crear estructuras de datos personalizadas que se ajusten a necesidades específicas. A continuación, te muestro cómo puedes implementar una cola básica desde cero, utilizando arreglos y listas enlazadas.

Implementación de una Cola Usando Arreglos
Código de Cola Estática
Aquí hay un ejemplo de cómo implementar una cola utilizando un arreglo (o array) estático:

```
#include <iostream>
#define MAX 5 // Capacidad máxima de la cola

using namespace std;

// Definición de la estructura Cola
struct Cola {
     int elementos[MAX]; // Array para almacenar los elementos
     int frente;            // Índice del frente
     int trasero;           // Índice del trasero
     int tamano;            // Número de elementos actuales en la cola

     // Constructor para inicializar la cola vacía
     Cola() {
          frente = 0;
          trasero = -1;
          tamano = 0;
     }

     // Verificar si la cola está vacía
     bool estaVacia() {
          return (tamano == 0);
     }

     // Verificar si la cola está llena
     bool estaLlena() {
          return (tamano == MAX);
     }

     // Enqueue: Insertar un elemento al final de la cola
     void encolar(int valor) {
          if (estaLlena()) {
               cout << "Error: La cola está llena" << endl;
               return;
          }
          trasero = (trasero + 1) % MAX; // Ciclo circular
          elementos[trasero] = valor;
          tamano++;
          cout << "Elemento " << valor << " encolado" << endl;
     }

     // Dequeue: Eliminar un elemento del frente de la cola
     void desencolar() {
          if (estaVacia()) {
               cout << "Error: La cola está vacía" << endl;
               return;
          }
          cout << "Elemento " << elementos[frente] << " desencolado" << endl;
          frente = (frente + 1) % MAX; // Ciclo circular
          tamano--;
     }

     // Obtener el elemento al frente sin desencolar
```

```
    int obtenerFrente() {
        if (estaVacia()) {
            cout << "Error: La cola está vacía" << endl;
            return -1;
        }
        return elementos[frente];
    }

    // Obtener el tamaño actual de la cola
    int obtenerTamano() {
        return tamano;
    }
};

int main() {
    Cola cola; // Crear una cola

    // Operaciones de encolar
    cola.encolar(10);
    cola.encolar(20);
    cola.encolar(30);
    cola.encolar(40);
    cola.encolar(50);

    // Intentar encolar en una cola llena
    cola.encolar(60);

    // Mostrar el frente y tamaño actual de la cola
    cout << "Elemento en el frente: " << cola.obtenerFrente() << endl;
    cout << "Tamaño actual: " << cola.obtenerTamano() << endl;

    // Operaciones de desencolar
    cola.desencolar();
    cola.desencolar();

    // Mostrar el nuevo frente y tamaño después de desencolar
    cout << "Elemento en el frente: " << cola.obtenerFrente() << endl;
    cout << "Tamaño actual: " << cola.obtenerTamano() << endl;

    return 0;
}
```

Explicación de la Implementación con Arreglos
Estructura Cola: Definimos una estructura llamada Cola que incluye un arreglo para almacenar los elementos, índices para el frente y el trasero, y el tamaño de la cola.
Ciclo Circular: La cola usa un enfoque de ciclo circular para aprovechar al máximo el espacio del arreglo, evitando así desperdiciar posiciones vacías.
Métodos: Implementamos métodos para encolar, desencolar, obtener el elemento al frente, verificar si la cola está vacía o llena, y obtener el tamaño actual.
Implementación de una Cola Usando Listas Enlazadas

Otra forma de implementar una cola es utilizando listas enlazadas. Esto es útil porque permite que la cola crezca y se reduzca dinámicamente.

Código de Cola Dinámica

```
#include <iostream>

using namespace std;

// Definición del nodo de la lista enlazada
struct Nodo {
    int dato;
    Nodo* siguiente;

    Nodo(int valor) : dato(valor), siguiente(nullptr) {}
};

// Definición de la estructura Cola
struct Cola {
    Nodo* frente; // Apunta al frente de la cola
```

```
    Nodo* trasero; // Apunta al final de la cola

    // Constructor para inicializar la cola vacía
    Cola() : frente(nullptr), trasero(nullptr) {}

    // Verificar si la cola está vacía
    bool estaVacia() {
        return (frente == nullptr);
    }

    // Enqueue: Insertar un elemento al final de la cola
    void encolar(int valor) {
        Nodo* nuevoNodo = new Nodo(valor);
        if (estaVacia()) {
            frente = nuevoNodo;
        } else {
            trasero->siguiente = nuevoNodo;
        }
        trasero = nuevoNodo;
        cout << "Elemento " << valor << " encolado" << endl;
    }

    // Dequeue: Eliminar un elemento del frente de la cola
    void desencolar() {
        if (estaVacia()) {
            cout << "Error: La cola está vacía" << endl;
            return;
        }
        Nodo* temp = frente;
        cout << "Elemento " << frente->dato << " desencolado" << endl;
        frente = frente->siguiente;
        delete temp; // Liberar memoria
        if (frente == nullptr) {
            trasero = nullptr; // Si la cola queda vacía, actualizar trasero
        }
    }

    // Obtener el elemento al frente sin desencolar
    int obtenerFrente() {
        if (estaVacia()) {
            cout << "Error: La cola está vacía" << endl;
            return -1;
        }
        return frente->dato;
    }
};

int main() {
    Cola cola; // Crear una cola

    // Operaciones de encolar
    cola.encolar(10);
    cola.encolar(20);
    cola.encolar(30);

    // Mostrar el frente
    cout << "Elemento en el frente: " << cola.obtenerFrente() << endl;

    // Desencolar un elemento
    cola.desencolar();
    cout << "Elemento en el frente después de desencolar: " << cola.obtenerFrente() <<
endl;

    // Desencolar todos los elementos
    cola.desencolar();
    cola.desencolar();
    cola.desencolar(); // Intentar desencolar de una cola vacía

    return 0;
}
```

Explicación de la Implementación con Listas Enlazadas
Estructura de Nodo: Cada nodo contiene un dato y un puntero al siguiente nodo.
Cola Dinámica: La cola tiene punteros al frente y al trasero. Cuando se encola un elemento, se crea un nuevo nodo y se añade al final de la lista.
Desencolar: Cuando se desencola un elemento, se elimina el nodo del frente y se actualiza el puntero al frente.

Conclusión
Ambas implementaciones, ya sea utilizando arreglos o listas enlazadas, permiten trabajar con colas en C++ sin depender de la biblioteca STL. Cada enfoque tiene sus ventajas y desventajas. Las colas basadas en arreglos son más simples, pero tienen una capacidad fija, mientras que las colas basadas en listas enlazadas son más flexibles y pueden crecer dinámicamente.

Implementación Biblioteca STL.
Una cola es una estructura de datos en la que los elementos se insertan por un extremo (llamado final o trasero de la cola) y se eliminan por el otro extremo (llamado frente o delantero de la cola). Este comportamiento se conoce como **FIFO** (First In, First Out), *lo que significa que el primer elemento en entrar es el primero en salir*.

Funciones principales de una Cola:
Enqueue (inserción): Agregar un elemento al final de la cola.
Dequeue (eliminación): Eliminar un elemento desde el frente de la cola.
Front: Obtener el elemento en el frente sin eliminarlo.
Empty: Verificar si la cola está vacía.
Size: Obtener el número de elementos en la cola.

Implementación de una Cola en C++
Puedes implementar una cola en C++ utilizando:

Estructuras (struct).
La biblioteca estándar STL (std::queue).
Vamos a centrarnos en la implementación manual usando estructuras.

Implementación de una Cola con struct
Esta implementación será una cola estática simple utilizando un array y una estructura.

Código:

```
#include <iostream>
#define MAX 5 // Capacidad máxima de la cola

using namespace std;

// Definición de la estructura Cola
struct Cola {
     int elementos[MAX]; // Array para almacenar los elementos
     int frente;            // Índice del frente
     int trasero;           // Índice del trasero
     int tamano;            // Número de elementos actuales en la cola

     // Constructor para inicializar la cola vacía
     Cola() {
          frente = 0;
          trasero = -1;
          tamano = 0;
     }

     // Verificar si la cola está vacía
     bool estaVacia() {
          return (tamano == 0);
     }

     // Verificar si la cola está llena
     bool estaLlena() {
          return (tamano == MAX);
     }

     // Enqueue: Insertar un elemento al final de la cola
     void encolar(int valor) {
          if (estaLlena()) {
```

```
            cout << "Error: La cola está llena" << endl;
            return;
        }
        trasero = (trasero + 1) % MAX; // Ciclo circular
        elementos[trasero] = valor;
        tamano++;
        cout << "Elemento " << valor << " encolado" << endl;
    }

    // Dequeue: Eliminar un elemento del frente de la cola

    void desencolar() {
        if (estaVacia()) {
            cout << "Error: La cola está vacía" << endl;
            return;
        }
        cout << "Elemento " << elementos[frente] << " desencolado" << endl;
        frente = (frente + 1) % MAX; // Ciclo circular
        tamano--;
    }

    // Obtener el elemento al frente sin desencolar
    int obtenerFrente() {
        if (estaVacia()) {
            cout << "Error: La cola está vacía" << endl;
            return -1;
        }
        return elementos[frente];
    }

    // Obtener el tamaño actual de la cola
    int obtenerTamano() {
        return tamano;
    }
};

int main() {
    Cola cola; // Crear una cola

    // Operaciones de encolar
    cola.encolar(10);
    cola.encolar(20);
    cola.encolar(30);
    cola.encolar(40);
    cola.encolar(50);

    // Intentar encolar en una cola llena
    cola.encolar(60);

    // Mostrar el frente y tamaño actual de la cola
    cout << "Elemento en el frente: " << cola.obtenerFrente() << endl;
    cout << "Tamaño actual: " << cola.obtenerTamano() << endl;

    // Operaciones de desencolar
    cola.desencolar();
    cola.desencolar();

    // Mostrar el nuevo frente y tamaño después de desencolar
    cout << "Elemento en el frente: " << cola.obtenerFrente() << endl;
    cout << "Tamaño actual: " << cola.obtenerTamano() << endl;

    return 0;
}
```

Explicación del Código
Definición de la Estructura Cola:

Tiene un array elementos para almacenar los elementos de la cola.
Los índices frente y trasero se usan para rastrear el frente y el final de la cola, respectivamente.
El atributo tamano se utiliza para llevar la cuenta del número de elementos en la cola.

Los métodos encolar y desencolar son para insertar y eliminar elementos de la cola.
Método encolar:

Verifica si la cola está llena. Si no lo está, agrega el nuevo elemento al final y actualiza el índice trasero de manera circular.
Método desencolar:

Verifica si la cola está vacía. Si no lo está, elimina el elemento al frente y ajusta el índice frente de manera circular.
Método obtenerFrente:

Devuelve el valor del frente sin eliminarlo.

Método obtenerTamano:

Devuelve el tamaño actual de la cola.
Ejemplo de Ejecución

```
Elemento 10 encolado
Elemento 20 encolado
Elemento 30 encolado
Elemento 40 encolado
Elemento 50 encolado
Error: La cola está llena
Elemento en el frente: 10
Tamaño actual: 5
Elemento 10 desencolado
Elemento 20 desencolado
Elemento en el frente: 30
Tamaño actual: 3
```

Implementación con la Biblioteca Estándar

Si deseas usar la STL de C++, puedes simplificar mucho la implementación usando `std::queue`. Aquí un ejemplo:

```
#include <iostream>
#include <queue>

using namespace std;

int main() {
     queue<int> cola;

     // Encolar elementos
     cola.push(10);
     cola.push(20);
     cola.push(30);

     // Mostrar el frente
     cout << "Elemento en el frente: " << cola.front() << endl;

     // Desencolar un elemento
     cola.pop();
     cout << "Elemento en el frente después de desencolar: " << cola.front() << endl;

     // Tamaño de la cola
     cout << "Tamaño actual: " << cola.size() << endl;

     return 0;
}
```

Conclusión
Las colas son estructuras de datos fundamentales que siguen el principio FIFO. Pueden implementarse de manera manual con arrays y estructuras o usar la implementación de la STL que facilita su uso. Ambas opciones son válidas dependiendo del contexto y necesidades del proyecto.