Resumen de Programación en C++ por Semanas

Semana N° 1 - Introducción y Repositorio de Código

Descripción: Se introduce al lenguaje C++, su sintaxis básica y el uso de repositorios de código (como GitHub) para el control de versiones y colaboración.

Ventajas:

- Facilita la comprensión de la estructura de un programa.
- El uso de repositorios fomenta la colaboración y el control de versiones.

Desventajas:

- Puede ser confuso para principiantes manejar herramientas como Git.
- La sintaxis de C++ puede parecer compleja al inicio.

Ejemplo 1:

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

int main() {
    cout << "Hola, mundo!" << endl;
    return 0;
}
```

Ejemplo 2:

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

int main() {
    int edad = 20;
    cout << "Tienes " << edad << " a os." << endl;
    return 0;
}
```

Semana N° 2 - Arrays

Descripción: Se estudia el uso de arreglos para almacenar múltiples valores del mismo tipo.

Ventajas:

- Acceso rápido a los elementos por índice.
- Útil para manejar grandes cantidades de datos homogéneos.

Desventajas:

- Tamaño fijo: no se puede redimensionar.
- No hay verificación de límites por defecto (puede causar errores).

Ejemplo 1:

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

int main() {
    int numeros[5] = {1, 2, 3, 4, 5};
    for(int i = 0; i < 5; i++) {
        cout << numeros[i] << " ";
    }
    return 0;
}
```

Ejemplo 2:

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

int main() {
    int nums[4] = {10, 20, 30, 40}, suma = 0;
    for(int i = 0; i < 4; i++) {
        suma += nums[i];
    }
    cout << "Suma: " << suma << endl;
    return 0;
}
```

Semana N° 3 - Structs

Descripción: Permite definir estructuras que agrupan diferentes tipos de datos.

Ventajas:

- Organiza mejor los datos.
- Útil para representar objetos reales.

Desventajas:

- No soporta funciones miembro como las clases.
- Puede volverse complejo con estructuras anidadas.

Ejemplo 1:

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

struct Persona {
    string nombre;
    int edad;
};

int main() {
    Persona p = {"Juan", 25};
    cout << p.nombre << " tiene " << p.edad << " a os." << endl;
    return 0;
}
```

Ejemplo 2:

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

struct Libro {
    string titulo;
    int paginas;
};

int main() {
    Libro libros[2] = {{"C++ B sico", 300}, {"Estructuras", 250}};
    for(int i = 0; i < 2; i++) {
        cout << libros[i].titulo << ": " << libros[i].paginas << "
           p ginas\n";
    }
    return 0;
}
```

Semana N° 4 - Listas Enlazadas

Descripción: Estructura dinámica donde cada nodo apunta al siguiente.
Ventajas:

- Inserciones y eliminaciones eficientes.
- Tamaño dinámico.

Desventajas:

- Acceso solo secuencial.

- Requiere más memoria por los punteros.

Ejemplo 1:

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

struct Nodo {
    int dato;
    Nodo* siguiente;
};

int main() {
    Nodo* cabeza = new Nodo{10, nullptr};
    cabeza->siguiente = new Nodo{20, nullptr};

    Nodo* temp = cabeza;
    while(temp) {
        cout << temp->dato << " -> ";
        temp = temp->siguiente;
    }
    return 0;
}
```

Ejemplo 2:

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

struct Nodo {
    int dato;
    Nodo* siguiente;
};

void insertar(Nodo*& cabeza, int valor) {
    Nodo* nuevo = new Nodo{valor, cabeza};
    cabeza = nuevo;
}

int main() {
    Nodo* lista = nullptr;
    insertar(lista, 30);
    insertar(lista, 20);
    insertar(lista, 10);

    Nodo* temp = lista;
    while(temp) {
        cout << temp->dato << " ";
        temp = temp->siguiente;
    }
    return 0;
}
```

Semana N° 5 - Listas Dobles y Circulares

Descripción: Extensiones de las listas enlazadas, con doble enlace o conexión circular.
Ventajas:

- Listas dobles: recorrido hacia atrás y adelante.
- Listas circulares: recorrido continuo.

Desventajas:

- Mayor complejidad y uso de punteros.
- Más uso de memoria.

Ejemplo 1: Lista doblemente enlazada

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

struct Nodo {
    int dato;
    Nodo* anterior;
    Nodo* siguiente;
};

int main() {
    Nodo* primero = new Nodo{10, nullptr, nullptr};
    Nodo* segundo = new Nodo{20, primero, nullptr};
    primero->siguiente = segundo;

    Nodo* temp = primero;
    while(temp) {
        cout << temp->dato << " ";
        temp = temp->siguiente;
    }
    return 0;
}
```

Ejemplo 2: Lista circular

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

struct Nodo {
    int dato;
    Nodo* siguiente;
};

int main() {
    Nodo* nodo1 = new Nodo{10, nullptr};
    Nodo* nodo2 = new Nodo{20, nullptr};
    nodo1->siguiente = nodo2;
    nodo2->siguiente = nodo1;
```

```
    Nodo* temp = nodo1;
    int count = 0;
    do {
        cout << temp->dato << " ";
        temp = temp->siguiente;
        count++;
    } while(temp != nodo1 && count < 5);
    return 0;
}
```

Semana N° 6 - Colas (Queues)

Descripción: Estructura FIFO: el primero en entrar es el primero en salir.
Ventajas:

- Útil para modelar procesos, tareas y flujos.
- Operaciones simples.

Desventajas:

- Acceso restringido.
- Difícil de manejar sin estructuras auxiliares.

Ejemplo 1: Cola con STL

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

int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);

    while(!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    return 0;
}
```

Ejemplo 2: Cola personalizada

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

struct Nodo {
    int dato;
    Nodo* siguiente;
```

```
};

struct Cola {
    Nodo* frente = nullptr;
    Nodo* final = nullptr;

    void encolar(int valor) {
        Nodo* nuevo = new Nodo{valor, nullptr};
        if(final) final->siguiente = nuevo;
        else frente = nuevo;
        final = nuevo;
    }

    void desencolar() {
        if(frente) {
            Nodo* temp = frente;
            frente = frente->siguiente;
            delete temp;
        }
        if(!frente) final = nullptr;
    }

    void mostrar() {
        Nodo* temp = frente;
        while(temp) {
            cout << temp->dato << " ";
            temp = temp->siguiente;
        }
    }
};

int main() {
    Cola c;
    c.encolar(1);
    c.encolar(2);
    c.mostrar();
    return 0;
}
```

Semana N° 7 - Pilas (Stacks)

Descripción: Estructura LIFO: el último en entrar es el primero en salir.
Ventajas:

- Ideal para retroceso, expresiones y recursión.
- Implementación sencilla.

Desventajas:

- Solo acceso al tope.

- Puede causar desbordamientos.

Ejemplo 1: Pila con STL

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

int main() {
    stack<int> pila;
    pila.push(5);
    pila.push(10);

    while(!pila.empty()) {
        cout << pila.top() << " ";
        pila.pop();
    }
    return 0;
}
```

Ejemplo 2: Verificar paréntesis

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

bool balanceado(string exp) {
    stack<char> pila;
    for(char c : exp) {
        if(c == '(') pila.push(c);
        else if(c == ')') {
            if(pila.empty()) return false;
            pila.pop();
        }
    }
    return pila.empty();
}

int main() {
    string exp = "(a+b)*(c-d)";
    cout << (balanceado(exp) ? "Balanceado" : "No balanceado");
    return 0;
}
```

Semana N° 8 - Recursión

Descripción: Una función que se llama a sí misma para resolver subproblemas.
Ventajas:

- Código limpio y natural para muchos algoritmos.
- Útil para estructuras jerárquicas.

Desventajas:

- Uso de pila de llamadas.
- Puede causar errores si no se define bien el caso base.

Ejemplo 1: Factorial

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

int factorial(int n) {
    return (n == 0) ? 1 : n * factorial(n - 1);
}

int main() {
    cout << "5! = " << factorial(5) << endl;
    return 0;
}
```

Ejemplo 2: Fibonacci

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

int fibonacci(int n) {
    if(n <= 1) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}

int main() {
    for(int i = 0; i < 6; i++) {
        cout << fibonacci(i) << " ";
    }
    return 0;
}
```