Resumen detallado: Programmierpraktikum C und C++ - TU Darmstadt

Profesor: Dr.-Ing. Eric Lenz Cátedra: Control and Cyber-Physical Systems (CCPS) Universidad: TU Darmstadt Fecha: 2025

## **INTRODUCCIÓN**

Este curso es un "Programmierpraktikum", es decir, una práctica intensiva en programación orientada a la enseñanza de los fundamentos y aspectos avanzados del lenguaje C++, con una introducción al lenguaje C en el contexto de sistemas embebidos. Se espera que los estudiantes tengan conocimientos básicos de programación, especialmente en Java.

El curso está diseñado en modalidad híbrida y se divide en dos grandes bloques:

1. **Teórico (online):** Introducción y profundización en conceptos de C++ y C.
2. **Práctico (presencial):** Aplicación sobre microcontroladores reales.

El contenido está distribuido en módulos temáticos identificados con etiquetas como [G] (Grundlagen), [S] (Speicherverwaltung), [O] (Objektorientierung), [F] (Fortgeschrittene Themen), [C] (Einführung in C), [Z] (Zusatzaufgaben).

## **1. BLOQUE [G] - FUNDAMENTOS DE C++**

### **1.1 Estructura del Proyecto en C++**

* Archivos .hpp (cabeceras) contienen las declaraciones de clases y funciones.
* Archivos .cpp contienen la implementación.
* Uso de #include para incluir dependencias.
* Recomendación: separar lógica en múltiples archivos para mantener modularidad.

### **1.2 Compilación en C++**

El proceso consta de tres fases:

* **Compile Time:** cada archivo .cpp se traduce a un archivo objeto .o.
* **Link Time:** los archivos .o se enlazan entre sí.
* **Load Time:** el sistema operativo carga y ejecuta el programa.

### **1.3 Comparación Java vs C++**

* Java usa una JVM para interpretar bytecode, mientras que C++ compila directamente a binario.
* En C++, la gestión de dependencias y linking es explícita.

### **1.4 Include Guards y #pragma once**

* Los #ifndef, #define y #endif protegen contra inclusiones múltiples.
* #pragma once es más simple, aunque no estándar. La mayoría de compiladores modernos lo soportan.

### **1.5 Declaración vs Definición**

* **Declarar**: informar que algo existe (ej. class A; o void f();).
* **Definir**: proporcionar la implementación (ej. class A { ... };).

### **1.6 Undefined Behavior (UB)**

Errores como:

* Acceder a punteros nulos.
* Dividir por cero.
* No retornar valor en funciones.
* Usar variables no inicializadas. Estos errores pueden resultar en comportamiento impredecible, ya que el compilador puede asumir que nunca ocurren.

### **1.7 Const Correctness**

Uso de const para garantizar que una variable no sea modificada. Ejemplo:

const int x = 42;

int const \*ptr = &x; // Puntero a constante

int \*const ptr2 = &y; // Puntero constante

### **1.8 Tipos de Variables**

* **Wert-Typ:** contiene valor directamente (int x = 3;).
* **Referenz-Typ:** alias de otro valor (int &r = x;).
* **Pointer-Typ:** dirección de memoria (int \*p = &x;).
* **Array-Typ:** puntero a memoria contigua (int arr[] = {1,2,3};).

## **2. BLOQUE [S] - GESTIÓN DE MEMORIA**

### **2.1 Zonas de memoria**

* **Text:** código binario del programa.
* **Data/BSS:** variables globales (inicializadas y no).
* **Stack:** variables locales, parámetros de función.
* **Heap:** memoria dinámica manejada con new y delete.

### **2.2 Stack vs Heap**

* Stack: rápido, automático, limitado.
* Heap: manual, más grande, lento.

### **2.3 Punteros**

* Variables que contienen direcciones de otras variables.
* Se usa & para obtener la dirección, \* para acceder al valor.

### **2.4 Null Pointer**

int\* ptr = nullptr; // C++11

### **2.5 sizeof y size\_t**

* sizeof() devuelve tamaño en bytes.
* std::size\_t es el tipo estándar para almacenar tamaños.

### **2.6 Const y punteros**

Diferencias entre:

* const int\* p – valor apuntado no puede cambiar.
* int\* const p – puntero no puede cambiar, valor sí.
* const int\* const p – ninguno puede cambiar.

### **2.7 Referencias**

* Alias de una variable.
* Sintaxis: int& r = x;

## **3. BLOQUE [O] - PROGRAMACIÓN ORIENTADA A OBJETOS (POO)**

### **3.1 class vs struct vs union**

* class: miembros privados por defecto.
* struct: miembros públicos por defecto. Se hereda de C.
* union: todos los miembros comparten la misma dirección de memoria. Útil para estructuras alternativas.

### **3.2 Encapsulamiento y visibilidad**

* public: accesible desde fuera de la clase.
* protected: accesible desde la clase y sus derivados.
* private: accesible solo desde la clase.
* friend: una función o clase externa puede acceder a miembros privados.

### **3.3 Métodos**

* Constructores: inicializan objetos.
* Destructores: liberan recursos (~Clase())
* Métodos constantes: int getX() const;

### **3.4 Referencias a objetos**

Building b;

Building& ref = b;

### **3.5 Sobrecarga de operadores**

* Operadores como +, =, [] pueden redefinirse.

Vector operator+(const Vector& other);

### **3.6 Namespaces**

* Ayudan a evitar conflictos de nombres.

namespace utils {

void print();

}

## **4. BLOQUE [F] - TEMAS AVANZADOS**

### **4.1 Templates**

* Permiten definir funciones y clases genéricas.

template<typename T>

T add(T a, T b) { return a + b; }

### **4.2 Functores y punteros a función**

typedef int(\*funcPtr)(int, int);

int add(int a, int b) { return a + b; }

### **4.3 Lambdas (desde C++11)**

auto lambda = [](int a, int b) { return a + b; };

## **5. BLOQUE [C] - C Y SISTEMAS EMBEBIDOS**

### **5.1 Introducción a C**

* Lenguaje procedural.
* Sin clases, solo structs.
* Uso explícito de punteros.

### **5.2 Aplicación en Microcontroladores**

* C se usa por su eficiencia y control sobre el hardware.
* Acceso directo a memoria, registros, puertos.

### **5.3 Ejemplo: Blinking LED**

#include <avr/io.h>

#include <util/delay.h>

int main(void) {

DDRB |= (1 << DDB5);

while (1) {

PORTB ^= (1 << PORTB5);

\_delay\_ms(1000);

}

}

## **6. BLOQUE [Z] - BONUS Y EJERCICIOS**

### **6.1 Bonus**

* Cada bloque tiene una tarea adicional.
* Puntaje acumulativo puede mejorar la nota final hasta 1.0.
* No ayudan a aprobar, solo mejoran la nota.

### **6.2 Aufzugsimulation (Simulador de Ascensor)**

* Proyecto integrador.
* Incluye múltiples clases (Building, Elevator, Floor).
* Diseño orientado a objetos, estrategias, métricas (energía, tiempo de espera).

## **7. HERRAMIENTAS RECOMENDADAS**

* CLion IDE (con soporte para CMake y GDB).
* OnlineGDB, cpp.sh, godbolt.org para pruebas rápidas.
* Referencias: cppreference.com, learncpp.com, The C Book, Effective C++.

## **8. EVALUACIÓN Y EXAMEN**

* **Klausur** el 26/09/2025.
* Duración: 90 minutos.
* Temas: Todo excepto los marcados como [Exkurs] u opcionales.
* Sin calculadora, lápiz ni corrector.