UNIVERSIDAD NACIONAL DE HURLINGHAM

Universidad Nacional de San Martín

Introduction

Leandro Luciano Gagliardi
lgagliardi@unsam.edu.ar

¿Qué es el CDAC?

El Centro para el Desarrollo de la Computación Avanzada (C-DAC) es la principal organización de I+D del Ministerio de Electrónica y Tecnología de la Información (MeitY) para llevar a cabo I+D en TI, electrónica y áreas asociadas. Las diferentes áreas del C-DAC se originaron en diferentes momentos, muchas de las cuales surgieron como resultado de la identificación de oportunidades.

Programa

1. Basics of Program Writing & Coding Practices
2. Overview of C Programming language
3. Introduction to GNU Toolchain and GNU Make utility
4. Linux environment and vi editor
5. Tokens of C - Keywords, Data-Types, Variables, Constants, Operators, Identifiers
6. Storage Class Specifiers
7. Control Flow Statements
8. Arrays, Multidimensional arrays, Data Input & Output, Strings, Loops - for, while etc.
9. Functions and Recursion
10. Pointers - Introduction, Pointer Arithmetic, Pointers and Arrays, Pointers and Functions,
11. Pointers and Strings
12. Structures, Unions, Enum, Typedef, Bit field operators and pointers with structures
13. Preprocessors, C and Assembly, Files, I/O,
14. Variable number of arguments, Command Line arguments, Error handling
15. Debugging and Optimization of C programs
16. Bit operations & Handling portability issues in C
17. Hardware, Time, Space and Power aware Programming
18. Secure Programming in Embedded C – tools and techniques

¿Qué son los sistemas embebidos?

- Un sistema embebido es un sistema informático de propósito especial.
- Está completamente encapsulado por el dispositivo que controla.
- Un sistema embebido tiene requerimientos específicos y realiza tareas predefinidas.
- Un sistema embebido es un dispositivo de hardware programado.
- Un chip de hardware programable es la “materia prima”.
- Programado con aplicaciones específicas.
- El software controla la operación y la funcionalidad.

Un sistema embebido es un sistema de control en tiempo real, controlado por software y basado en microcontrolador, que puede ser autónomo o interactivo con humanos o redes, y que opera en diversas variables físicas y en diversos entornos.

¿Donde encontrar Sistemas Embebidos?

- Electrodomésticos: Microondas, lava vajillas
- Entretenimiento: Televisores, Reproductores de DVD/BlueRay, MP3, Consolas de videojuegos.
- Telecomunicaciones: Teléfonos celulares, fijos.
- Computadoras: teclados, discos externos.
- Vehículos: computadora de abordo, tablero, sistema de inyección, sistema de suspensión, GPS.

Características de los Sistemas Embebidos

- La función principal es el control y no la computación.
- Usualmente diseñado para realizar funciones específicas a un bajo costo.
- El microcontrolador o microprocesador está embebido.
- El firmware está embebido en los dispositivos de hardware.
- Los programas deben ejecutarse con restricciones de tiempo real y con recursos de hardware limitados.
- Los sistemas embebidos residen en máquinas, de las cuales se espera un funcionamiento de forma continua durante años sin errores.
- El software y el firmware suelen desarrollarse y probarse con más cuidado que el software para computadoras personales.
- Algunos sistemas embebidos pueden estar fuera del alcance de los humanos.

Características de los Sistemas Embebidos

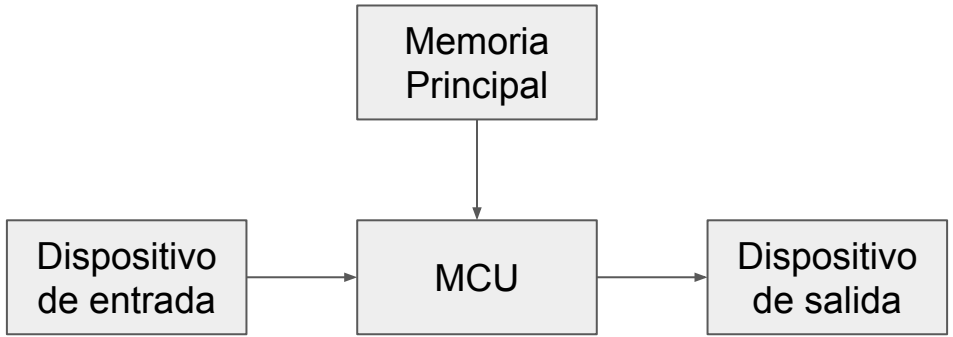
- Componentes: hardware y software.
- El controlador debe responder rápidamente para mantener la operación dentro de una región segura.
- Operar en tiempo real.
- Interacción entre subsistemas.
- Capacidad de conectarse a Internet.
- Los productos no pueden permitirse el lujo de tener errores.
- La actualización no es sencilla.
- La reputación está en juego.

Subsistemas

Tres subsistemas principales

- Microcontrolador
- Memoria
  - Para almacenar programas
  - Para almacenar datos
- Entrada y salida

Elementos de interconexión (BUS)



UNIVERSIDAD NACIONAL DE HURLINGHAM

Universidad Nacional de San Martín

Basics of Program Writing & Coding Practices

Leandro Gagliardi
lgagliardi@unsam.edu.ar

Lenguajes de bajo nivel

En esta categoría podemos encontrar dos: **lenguaje de máquina** y **assembler**.

- **Lenguaje de máquina**: las computadoras sólo pueden entender instrucciones en código binario, es decir, **0** y **1**. Sin embargo, un programa en **lenguaje de máquina** es propenso a errores y su mantenimiento es muy difícil. Además, los programas en **lenguaje de máquina** no son portables. Cada ordenador tiene sus propias instrucciones de máquina, por lo que los programas escritos para un ordenador no son válidos para otros ordenadores.
- **Assembler**: una forma modificada de **lenguaje de máquina**. En **assembler**, las instrucciones se dan en inglés, en palabras como **MOV**, **ADD**, **SUB**, etc. Por lo tanto, es más fácil escribir y comprender programas en **assembler**. Dado que una computadora solo puede comprender **lenguaje de máquina**, los programas en **assembler** deben traducirse. El traductor que se utiliza para se llama "**assembler**". Aunque escribir programas en **assembler** es un poco más fácil, el programador debe conocer todos los detalles de bajo nivel relacionados con el hardware de una computadora. El programa en **assembler** tampoco es portable. Dado que los lenguajes de bajo nivel están cerca del hardware, su ejecución es más rápida.

Lenguajes de alto nivel

Los lenguajes de alto nivel están diseñados teniendo en cuenta la portabilidad, es decir, estos lenguajes son independientes de la máquina. Al programar en un lenguaje de alto nivel, el programador no se preocupa por los detalles de bajo nivel, por lo que toda la atención se puede prestar a la lógica del problema que se está resolviendo. Para traducir un programa de lenguaje de alto nivel a lenguaje de máquina, se utiliza un **compilador**. Cada lenguaje tiene su propio **compilador**. Algunos lenguajes de esta categoría son:

- **FORTRAN:** orientado a computación de alto rendimiento en aplicaciones científicas y matemáticas.
- **COBOL:** orientado a los negocios y la administración.
- **BASIC: B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode.
- **Pascal:** orientado a la educación y posteriormente usado para desarrollo de software.

Algunos consejos antes de comenzar

1. **Nombres Significativos:** Usar nombres descriptivos para variables y funciones que indiquen su propósito o uso. Ejemplo: en lugar de usar **a**, **b**, **c**, usa **nombreUsuario**, **precioProducto**, **fechaNacimiento**.
2. **Funciones Cortas:** Dividir el código en funciones que realicen una única tarea. Evitar funciones largas; si una función hace más de una cosa, considerar la posibilidad de dividirla.
3. **Consistencia:** Mantener un estilo de código consistente en todo el proyecto (nombres de variables, indentación, etc.).
4. **Comentarios Claros y Útiles:** Comentar el código cuando sea necesario para explicar. Evitar comentarios redundantes que simplemente describen lo que ya es obvio en el código.
5. **Evita la Duplicación de Código (DRY - Don't Repeat Yourself):** Usar funciones para eso.
6. **Control de Versiones:** Usar Git y realizar commits frecuentes con mensajes descriptivos.
7. **Manejo de Errores:** Proporcionar mensajes de error que ayuden a diagnosticar el problema.
8. **Documentación:** Documentar el código, especialmente si es parte de una biblioteca o API.
9. **Minimizar el Uso de Globales:** Evitar el uso de variables globales que puedan afectar el comportamiento del programa en diferentes partes del código.

UNIVERSIDAD NACIONAL DE HURLINGHAM

Universidad Nacional de San Martín

Extra Slides

Leandro Gagliardi
lgagliardi@unsam.edu.ar