

 <b>UFAM</b>	<b>Universidade do Amazonas</b> <b>Faculdade de Tecnologia</b> <b>Departamento de Eletrônica e Computação</b> <b>Engenharia da Computação</b> <b>Programação de Tempo Real</b> <b>Prof. André Cavalcante</b>
--	---

## Trabalho 1: Ambiente de Desenvolvimento

**Aluno: MATHEUS ROCHA CANTO**

### OBJETIVOS

O objetivo deste laboratório foi a criação e validação de um ambiente de desenvolvimento robusto para a linguagem C em um Linux, sendo incluídos:

- Estruturar o projeto de forma modular, com diretórios apropriados para código-fonte, cabeçalhos e arquivos compilados.
- Desenvolver um **Makefile** para automatizar o processo de compilação e limpeza do projeto.
- Implementar e testar dois Tipos Abstratos de Dados (ADTs): um para manipulação de matrizes (**Matrix**) e outro para cálculo de integração numérica (**Integral**).

## INTRODUÇÃO TEÓRICA

O desenvolvimento de software em C em ambientes Unix-like, como o Linux, baseia-se em um conjunto de ferramentas clássicas e poderosas ao uso do compilador GCC (GNU Compiler Collection), sendo o padrão para traduzir o código-fonte C em código de máquina. Para projetos com múltiplos arquivos, a compilação manual torna-se ineficiente e propensa a erros onde a ferramenta `make` resolve esse problema ao ler um arquivo através do `Makefile`, que define as dependências entre os arquivos e os comandos necessários para construir o executável final, recompilando apenas o que foi modificado. A modularização é alcançada através da criação de Tipos Abstratos de Dados (ADTs), que separam a interface (`.h`) da implementação (`.c`), promovendo o encapsulamento e a reutilização de código.

## DESCRIÇÃO DOS ARQUIVOS FONTES

A modularização do projeto foi feita através da separação entre interface e implementação nos seguintes arquivos:

- **`matrix.h`**: Define a interface pública da ADT Matrix, incluindo a `struct Matrix` e os protótipos de suas funções.
- **`matrix.c`**: Contém a implementação das funções da ADT Matrix, como alocação de memória e algoritmos de operações matriciais.
- **`integral.h`**: Define a interface da ADT Integral, incluindo o tipo `Func` (ponteiro para função) e o protótipo da função de cálculo.
- **`integral.c`**: Implementação da função da ADT Integral, como implementação a Regra Composta do Trapézio.
- **`main.c`**: Atua como cliente dos ADTs, utilizando suas funções para realizar testes e validar o funcionamento do sistema.

## ESTRUTURA DA PASTA UTILIZADA

O projeto foi organizado na seguinte estrutura para garantir a separação de responsabilidades:

- **Trabalho 1 /** (Pasta Raiz);
  - **include/**: Contém os arquivos de cabeçalho (**.h**).
  - **src/**: Contém os arquivos de código-fonte (**.c**).
  - **Makefile**: Arquivo de automação da compilação.
  - **main**: Executável final, gerado na raiz do projeto.

## DESCRIÇÃO DO ARQUIVO DE COMPILAÇÃO (MAKEFILE)

O **Makefile** está estruturado em variáveis, regras e dependências.

- **Variáveis**: Permitem armazenar e reutilizar valores, tornando o ficheiro mais legível e fácil de manter.
  - **CC = gcc**: Define o compilador a ser utilizado.
  - **CFLAGS**: Contém as flags (opções) para o compilador, como **-Wall** (habilita avisos), **-g** (informações de depuração), **-std=c17** (padrão da linguagem) e **-Iinclude** (diretório de cabeçalhos).
  - **LDFLAGS**: Contém as flags para o linker, como **-lm** para incluir a biblioteca matemática.
- **Descoberta Automática**: Para evitar a necessidade de listar manualmente cada ficheiro-fonte, foram usadas funções do **make**:
  - **SOURCES = \$(wildcard src/\*.c)**: A função **wildcard** encontra todos os ficheiros que correspondem ao padrão **src/\*.c** e armazena a lista na variável **SOURCES**.

- `OBJECTS = $(patsubst src/%.c, obj/%.o, $(SOURCES))`: A função `patsubst` transforma a lista de ficheiros-fonte na lista de ficheiros-objeto correspondentes, que serão criados na pasta `obj`.
- **Regras:** Definem o que deve ser construído e como. A estrutura é `alvo : dependências.`
  - `all`: A regra principal que depende do executável final, desencadeando a compilação.
  - `$(TARGET) : $(OBJECTS)`: Regra de *linkagem*. Ela cria o executável final a partir de todos os ficheiros-objeto (`.o`).
  - `$(OBJDIR)%.o : $(SRCDIR)%.c`: Regra de *padrão* para a compilação. Ela ensina o `make` a criar um ficheiro `.o` em `obj/` a partir de um ficheiro `.c` correspondente em `src/`.
  - `clean` e `run`: Regras auxiliares para limpar o projeto e executar o programa, respetivamente.

## DESCRÍÇÃO DAS ADTs CRIADAS

### 1. ADT Matrix (.h e implementação)

O Tipo Abstrato de Dados `Matrix` foi desenvolvido para encapsular a representação e manipulação de matrizes bidimensionais de números de ponto flutuante (`double`).

- **Interface (`matrix.h`):** O ficheiro de cabeçalho define a estrutura `Matrix`, que armazena as dimensões (linhas e colunas) e um ponteiro duplo (`double **data`) para os dados alocados dinamicamente. Ele também declara os protótipos de todas as funções públicas, servindo como um "contrato" para o utilizador da biblioteca.

- **Implementação (`matrix.c`):** Contém a lógica interna das operações. As funções `create_matrix` e `free_matrix` geram o ciclo de vida da matriz, tratando da alocação e liberação de memória na heap. Foram implementadas funções para operações aritméticas básicas (soma, subtração, multiplicação por escalar e multiplicação de matrizes), bem como operações mais complexas como o cálculo da transposta, o determinante (implementado de forma recursiva através da expansão de Laplace) e a inversão de matrizes (utilizando o método da matriz adjugada).

## 2. ADT Integral (.h e implementação)

O Tipo Abstrato de Dados `Integral` foi criado para fornecer uma ferramenta genérica de cálculo de integração numérica definida.

- **Interface (`integral.h`):** A principal característica da interface é o uso de um ponteiro para função, definido pelo tipo `Func`. Isso permite que a função de integração aceite qualquer função matemática como argumento, desde que ela receba um `double` e retorne um `double`. Esta abordagem torna o ADT extremamente flexível e reutilizável.
- **Implementação (`integral.c`):** A implementação consiste na função `integral_trapezio`, que calcula a integral de uma dada função `f` num intervalo `[a, b]`. O método utilizado foi a Regra Composta do Trapézio, que aproxima a área sob a curva dividindo-a em `n` pequenos trapézios e somando as suas áreas. O método foi escolhido pela sua simplicidade de implementação e eficácia na implementação.

## **CONCLUSÃO**

A configuração do ambiente de desenvolvimento no Linux, utilizando GCC e Make, provou ser extremamente robusta e eficiente. A separação do código em `src` e `include` e a automação com o `Makefile` criam um fluxo de trabalho profissional e escalável. A experiência de construir e testar os ADTs reforçou a importância da gestão manual de memória em C e da criação de interfaces claras no ambiente Linux, sendo ideal para o desenvolvimento de sistemas complexos, como os que serão abordados na disciplina.