Componentes e Elementos de Programação: Variáveis e Tipos de Dados

Relatório Técnico Acadêmico | ADS 2026

Linguagem C | Alocação de Memória | Tipos Primitivos

1. Objetivos

Este relatório registra a execução prática dos conceitos de componentes e elementos de programação. O foco principal é a compreensão de como a linguagem C gerencia variáveis, constantes e tipos primitivos diretamente na memória do computador, utilizando compiladores para validar a alocação de dados.

---

2. Introdução Teórica

A programação estruturada exige que o desenvolvedor compreenda o ciclo de vida dos dados. Na linguagem C, as informações são alocadas em endereços físicos específicos, onde cada tipo de dado ocupa um espaço determinado em bytes.

- **Variáveis:** Espaços mutáveis na memória com tipo e endereço definidos.
- **Constantes:** Valores fixos definidos via `#define` ou `const` que não mudam durante a execução.
- **Endereçamento:** Acesso direto via operador `&`, essencial para a eficiência do sistema.

---

3. Ciclo de Vida e Escopo de Dados

A alocação de variáveis em C segue regras rígidas de escopo que determinam sua visibilidade e permanência na memória RAM:

- **Variáveis Locais:** Alocadas na *Stack* (pilha), existem apenas enquanto o bloco de código (função) está ativo.
- **Variáveis Globais:** Alocadas no segmento de dados, permanecem vivas durante toda a execução do programa.
- **Lixo de Memória:** Variáveis não inicializadas contêm valores residuais de acessos anteriores.

---

4. Metodologia

A atividade foi dividida em leitura teórica e experimentação em compilador. Utilizou-se o comando `sizeof` para validar o espaço físico ocupado por cada tipo de dado, além de testes com especificadores de formato (`%d`, `%f`, `%c`) nas funções de entrada e saída (`scanf` e `printf`).

O operador `sizeof()` na Linguagem C é uma ferramenta essencial para a gestão de memória. Ele não é uma função, mas um operador que retorna o tamanho, em bytes, que um determinado tipo de dado ou variável ocupa na memória RAM durante a execução do programa. No código implementado, o `sizeof()` foi utilizado para verificar a arquitetura do sistema, confirmando, por exemplo, que o tipo `int` aloca 4 bytes e o `double` aloca 8 bytes. Esta verificação é crucial para garantir a portabilidade do software e a eficiência no uso dos recursos do hardware.

Teste 1: Alocação de Memória

Objetivo: Validar o espaço físico (bytes) e o endereçamento lógico de variáveis primitivas.

```
# <<< Crie o arquivo aqui antês de rodar o comando! <<<

%%writefile memoria.c
#include <stdio.h>
#include <locale.h>

// Definição de cores para o terminal
#define RESET   "\033[0m"
#define VERDE   "\033[1;32m"
#define CIANO   "\033[1;36m"
#define LARANJA "\033[38;5;215m"

int main() {
    setlocale(LC_ALL, "Portuguese");

    // Identificação com cor Verde
    printf(VERDE "=========================================\n");
    printf("  RELATÓRIO TÉCNICO: ANÁLISE DE MEMÓRIA   \n");
    printf("  Acadêmico: Murilo Guimarães             \n");
    printf("=========================================\n\n" RESET);

    // Tabela com cor Ciano
    printf(CIANO "TABELA DE ALOCACAO (sizeof):\n");
    printf("-----------------------------------------\n" RESET);
    printf("Tipo CHAR:    " LARANJA "%zu byte\n" RESET, sizeof(char));
    printf("Tipo INT:     " LARANJA "%zu bytes\n" RESET, sizeof(int));
    printf("Tipo FLOAT:   " LARANJA "%zu bytes\n" RESET, sizeof(float));
    printf("Tipo DOUBLE:  " LARANJA "%zu bytes\n" RESET, sizeof(double));
    printf(CIANO "-----------------------------------------\n\n" RESET);

    // Exemplo de endereço com cor Laranja
    int variavelExemplo = 10;
    printf(VERDE "EXEMPLO DE ENDEREÇAMENTO:\n" RESET);
    printf("Valor da variável: " LARANJA "%d\n" RESET, variavelExemplo);
    printf("Endereço físico:   " LARANJA "%p\n" RESET, (void*)&variavelExemplo);
    printf(VERDE "=========================================\n" RESET);

    return 0;
}
```

Figura 1: Execução do código de alocação de memória em C no Colaboratory.

5. Comunicação via Especificadores

Para que o hardware interprete os bits corretamente, utilizamos especificadores de formato nas funções `printf` e `scanf`:

Tabela 1 - Especificadores de Formato em C

Especificador	Tipo Técnico	Finalidade
`%d`	Integer	Processamento de números inteiros decimais.
`%f`	Float	Representação de números reais de precisão simples.
`%c`	Char	Leitura de um único caractere (ASCII).
`%p`	Pointer	Exibição de endereço de memória hexadecimal.

Fonte: Elaborado pelo autor (2026).

6. Dados Obtidos e Análise

Abaixo, a síntese do espaço alocado para os tipos primitivos observados durante a prática, fundamentais para evitar o desperdício de memória em sistemas computacionais:

Tabela 2 - Alocação de Memória e Tipos de Dados em C

Tipo de Dado	Espaço (Bytes)	Uso Principal
`char`	1 byte	Caracteres únicos ou pequenos inteiros.
`int`	4 bytes	Números inteiros de precisão padrão.
`float`	4 bytes	Números reais (ponto flutuante).
`double`	8 bytes	Números reais de alta precisão.

Fonte: Elaborado pelo autor (2026).

Nota Técnica: *Cada tipo possui um limite. Por exemplo, um `int` de 4 bytes suporta valores até aproximadamente 2,1 bilhões. Tentar armazenar um valor maior causa o Integer Overflow, onde o número "capota" para o valor negativo mínimo. Veja esse e outros testes no Colaboratory, onde o código foi implementado e validado com acesso direto à RAM virtualizada.*

Execute com:

```
[3]
✓ 0s
# Compila o arquivo criado, gera um executável e executa o programa
!gcc memoria.c -o memoria && ./memoria

...
==========================================
  RELATÓRIO TÉCNICO: ANÁLISE DE MEMÓRIA
  Acadêmico: Murilo Guimarães
==========================================

TABELA DE ALOCACAO (sizeof):
------------------------------------------
Tipo CHAR:    1 byte
Tipo INT:     4 bytes
Tipo FLOAT:   4 bytes
Tipo DOUBLE:  8 bytes
------------------------------------------

EXEMPLO DE ENDEREÇAMENTO:
Valor da variável: 10
Endereço físico:   0x7ffccfaa79c4
==========================================
```

Figura 2: Resultado da execução no terminal validando a alocação e o endereçamento físico.

---

7. Resolução de Exercícios: "Faça Valer a Pena"

Questão 1: Tipos Primitivos

Análise: A questão aborda a correta declaração de variáveis para armazenar valores numéricos.

Resposta: Alternativa (c). Os valores numéricos podem ser guardados em tipos inteiros ou ponto flutuante, dependendo da necessidade de casas decimais.

Questão 2: Definição de Constantes

Resposta: Alternativa (a). O uso da diretiva #define permite criar constantes que funcionam como rótulos substituídos pelo pré-processador.

Questão 3: Asserções e Especificadores

- I: Incorreta (Uso de `%f` para variável que deveria ser inteira).
- II: Incorreta (Incompatibilidade de tipo e especificador).
- III: **Correta** (Uso correto de `%c` para o tipo `char`).

Resposta Final: Alternativa (d) - Apenas III está correta.

Conclusão

A execução desta prática permitiu consolidar a base teórica sobre a arquitetura de dados em C. Através do uso de `sizeof()` e do especificador `%p`, foi possível comprovar fisicamente que um `int` ocupa 4 bytes e visualizar o endereçamento direto na memória RAM. O domínio sobre a escolha correta dos tipos de dados é o que diferencia um código funcional de um software otimizado e profissional.

Referências Bibliográficas

ROVAI, K. R.; SCHEFFER, V. C.; ARTERO, M. A. **Algoritmos e programação estruturada**. Londrina: Editora e Distribuidora Educacional S.A., 2020. 192 p.

Este trabalho foi desenvolvido com o auxílio das ferramentas **NotebookLM** e **Google Gemini**. As IAs foram utilizadas como assistentes de produtividade para a estruturação do layout em HTML/CSS e para a validação técnica das explicações sobre o operador `sizeof()` e gestão de memória.
