UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
PROGRAMAÇÃO DE SISTEMAS DE TEMPO REAL
PROF. ANDRÉ CAVALCANTE
SEMESTRE 2024/2

LABORATÓRIO 1 - Preparação do Ambiente

Aluno: Matheus Carvalho Reges - 22152027

Este relatório descreve a implementação de três Tipos Abstratos de Dados (ADTs) em C: Dstring, Integral e Matrix. Cada um desses ADTs oferece um conjunto de operações que esconde os detalhes de baixo nível, permitindo que o programador se concentre na lógica de alto nível.

1. Dynamic String (Dstring) ADT

1.1 Objetivos

O objetivo deste trabalho é implementar um Tipo Abstrato de Dados (ADT) para manipulação de strings dinâmicas em C, chamado Dstring. O ADT oferece operações eficientes para criação, concatenação e gerenciamento de strings, superando as limitações das strings padrão da linguagem C.

1.2 Introdução Teórica

Um Tipo Abstrato de Dados (ADT) é uma representação que encapsula dados e define um conjunto de operações que podem ser realizadas sobre eles, permitindo que o programador interaja com esses dados sem se preocupar com os detalhes de implementação. A utilização de ADTs facilita a criação de programas mais organizados e modulares.
No contexto da linguagem C, onde a manipulação de strings é frequentemente limitada e propensa a erros devido ao gerenciamento manual de memória, o ADT Dstring se destaca como uma solução eficiente. O Dstring permite a criação e manipulação de strings de forma dinâmica, com alocação e liberação de memória automatizadas. Isso reduz o risco de estouros de buffer e simplifica operações como concatenação e cópia de strings, tornando o desenvolvimento mais seguro e eficiente. Ao abstrair a complexidade do gerenciamento de memória, o Dstring capacita os programadores a focar na lógica de alto nível, melhorando a produtividade e a qualidade do código.

1.3 Descrição dos Arquivos Fonte

- Dstring.h: Arquivo de cabeçalho que contém a definição da estrutura Dstring e as declarações das funções associadas.
- Dstring.c: Arquivo que implementa as funções definidas em Dstring.h.

- main.c: Arquivo que contém a função principal, onde o ADT Dstring é utilizado.

1.4 Descrição da estrutura de diretórios utilizada
Nesta estrutura, todos os arquivos relevantes estão localizados na mesma pasta, facilitando o acesso e a organização do projeto.

1.5 Descrição do arquivo de compilação (Makefile)
O Makefile apresentado automatiza o processo de compilação do projeto, simplificando a construção do executável a partir do código-fonte. Aqui estão os principais componentes do Makefile:

CC: O compilador a ser utilizado, que neste caso é o gcc (GNU Compiler Collection).

CFLAGS: As opções de compilação, que incluem -g para habilitar a geração de informações de depuração, -Wno-everything para desativar todos os avisos, -pthread para suporte a threads, e -lm para vincular a biblioteca matemática.

TARGET: O nome do executável gerado, definido como main.

Regras: O Makefile contém regras para compilar os arquivos fonte, utilizando a variável SRCS para listar todos os arquivos .c encontrados. A regra padrão (all) compila esses arquivos e gera o executável.

clean: Uma regra que remove os arquivos gerados, como o executável main e outros arquivos temporários, permitindo uma compilação limpa.

1.6 Descrição das ADTs criadas (.h e implementação)
- **Dstring.h**
    - char* buffer: Um ponteiro para o buffer de caracteres que armazena o conteúdo da string.
    - size_t length: Um campo que armazena o tamanho atual da string em bytes.

- As funções declaradas incluem:
    - Dstring* criarDstringPtr(const char* str): Cria uma nova Dstring a partir de uma string C.
    - Dstring* criarDstringChar(char c): Cria uma Dstring contendo um único caractere.
    - Dstring* criarDstringInt(int num): Cria uma Dstring a partir de um inteiro.
    - Dstring* criarDstringLong(long num): Cria uma Dstring a partir de um número longo.
    - Dstring* criarDstringFloat(float num): Cria uma Dstring a partir de um número em ponto flutuante.
    - Dstring* copiarDstring(const Dstring* dstr): Copia uma Dstring existente.
    - void concatenarDstrings(Dstring* dest, const Dstring* src): Concatena duas Dstrings, ajustando a memória conforme necessário.

- size_t obterTamanhoDstring(const Dstring* dstr): Retorna o comprimento da Dstring.
- char* obterBufferDstring(const Dstring* dstr): Retorna o buffer interno da Dstring.

- **Dstring.c**

O arquivo **Dstring.c** implementa as funções declaradas em Dstring.h. As principais características das funções implementadas são:

- **Alocação de memória**: Cada função de criação (criarDstringPtr, criarDstringChar, criarDstringInt, criarDstringLong, criarDstringFloat) aloca memória para a estrutura Dstring e seu buffer. Caso a alocação falhe, a função retorna NULL e libera a memória já alocada, evitando vazamentos.
- **Uso seguro de funções de string**: A função snprintf é utilizada para evitar estouros de buffer ao converter números para strings.
- **Concatenação e manipulação**: A função concatenarDstrings realoca o buffer de destino se necessário, garantindo que haja espaço suficiente para a nova string concatenada.
- **Verificações de segurança**: As funções obterTamanhoDstring e obterBufferDstring verificam se o ponteiro da Dstring é válido antes de acessar seus campos, evitando possíveis acessos inválidos à memória.
-

1.7 Conclusões sobre o ambiente de programação gerado

O ambiente de programação gerado oferece uma forma eficiente e segura de trabalhar com strings dinâmicas em C. A implementação do ADT Dstring promove a reutilização de código e a abstração dos detalhes de gerenciamento de memória. Além disso, o uso de um Makefile simplifica o processo de compilação, tornando-o acessível e organizado. Com estas ferramentas, é possível desenvolver aplicações que exigem manipulação flexível de strings, atendendo a diversas necessidades do programador.

2. Integral ADT

2.1 Objetivos

O objetivo do projeto é implementar diferentes métodos numéricos de integração em C, como a Soma de Riemann, Regra do Trapézio, Regra de Simpson, Regras de Quadratura, e Regra de Gauss-Legendre. Esses métodos permitem calcular aproximações para a integral de uma função definida em um intervalo específico, utilizando uma abordagem discreta. O foco está em criar um ambiente flexível para testes com várias funções e intervalos.

2.2 Introdução Teórica

A integração numérica é uma técnica usada para aproximar o valor de uma integral definida. Em muitas situações, encontrar a integral analítica de uma função é impossível ou inviável, então técnicas numéricas são usadas para obter uma aproximação aceitável. Neste projeto, as seguintes técnicas foram implementadas:

- **Soma de Riemann**: Dividindo o intervalo de integração em pequenas seções, o valor da função é calculado em pontos intermediários e somado para obter uma aproximação.

- **Regra do Trapézio**: Aproxima a área sob a curva utilizando trapézios em vez de retângulos, oferecendo uma melhor precisão do que a Soma de Riemann.
- **Regra de Simpson**: Usa parábolas para aproximar a curva da função, fornecendo resultados mais precisos em relação a métodos anteriores.
- **Quadratura**: Integração numérica que avalia a função em pontos médios de cada subintervalo.
- **Gauss-Legendre**: Método de quadratura mais sofisticado, que escolhe os pontos de avaliação (nós) e os pesos para minimizar o erro da aproximação.

2.3 Descrição dos Arquivos Fonte

O projeto é composto por três arquivos-fonte:

- **integral.h: Contém as declarações de funções e o typedef da função genérica usada em todas as integrações. Define os protótipos das funções de cada método numérico.**

- **integral.c: Implementa as funções declaradas no arquivo header, contendo as lógicas específicas para cada método numérico de integração (Soma de Riemann, Trapézio, Simpson, Quadratura e Gauss-Legendre).**

- **main.c: Contém a função main () e os casos de teste. Aqui são chamados os métodos de integração com uma função de exemplo (como sin(x)) e são exibidos os resultados.**

2.4 Descrição da Estrutura de Diretórios Utilizada

A estrutura do projeto é simples e direta, composta por três arquivos principais em um único diretório.

2.5 Descrição do Arquivo de Compilação (Makefile)

CC: O compilador a ser usado (no caso, gcc).
CFLAGS: As flags de compilação (neste caso, habilita warnings e otimizações de nível 2).
TARGET: O nome do executável a ser gerado.
Regras: Contém regras para compilar os arquivos objeto e linká-los para gerar o executável final.
clean: Remove os arquivos objeto e o executável.

2.6 Descrição das ADTs Criadas (.h e implementação)

• integral.h

- **typedef double (*func_ptr)(double x)**: Define um ponteiro de função que aceita um double como argumento e retorna um double. Isso permite passar diferentes funções matemáticas para os métodos de integração.
- As funções declaradas incluem:

- **double soma_riemann(func_ptr f, double inicio, double fim, int subdivisoes)**: Implementa a Soma de Riemann para integrar a função f no intervalo de inicio a fim, com um número especificado de subdivisões.
- **double regra_trapezio(func_ptr f, double inicio, double fim, int subdivisoes)**: Usa a Regra do Trapézio para calcular a integral da função f em um intervalo, utilizando subdivisões especificadas.
- **double regra_simpson(func_ptr f, double inicio, double fim, int subdivisoes)**: Aplica a Regra de Simpson para calcular a integral da função f com um número fixo de subdivisões.
- **double regra_quadratura(func_ptr f, double inicio, double fim, int subdivisoes)**: Implementa a Regra de Quadratura para aproximar a integral, avaliando a função nos pontos médios de cada subdivisão.
- **double regra_gauss_legendre(func_ptr f, double inicio, double fim, int subdivisoes)**: Utiliza a Regra de Gauss-Legendre para aproximar a integral da função, com dois pontos de Gauss em cada subdivisão.

• integral.c

O arquivo integral.c implementa as funções declaradas em integral.h. As principais características das funções implementadas são:

- **soma_riemann**: A função divide o intervalo em pequenos subintervalos de largura uniforme e soma os valores da função nos pontos iniciais de cada subintervalo para aproximar a integral.
- **regra_trapezio**: Esta função calcula a área de trapézios formados pela função em intervalos iguais. É uma melhoria em relação à Soma de Riemann, pois considera uma aproximação linear da função entre dois pontos adjacentes.
- **regra_simpson**: Usa uma combinação de retângulos e parábolas para fornecer uma aproximação muito precisa da integral, somando os valores ponderados da função em intervalos pares e ímpares.
- **regra_quadratura**: Avalia a função nos pontos médios de cada subintervalo, oferecendo uma aproximação equilibrada e precisa em muitos casos.
- **regra_gauss_legendre**: Utiliza uma técnica de quadratura que avalia a função em pontos de Gauss-Legendre, escolhidos para minimizar o erro da aproximação, fornecendo uma solução de alta precisão para polinômios de baixo grau.
- **Verificações de Segurança**: Cada função realiza verificações para garantir que a função de entrada seja válida e que o número de subdivisões não seja zero ou negativo, evitando comportamento indefinido.

2.7 Conclusões sobre o Ambiente de Programação Gerado

Em resumo, o projeto implementa cinco métodos numéricos de integração em C, com foco na flexibilidade e eficiência. A separação entre o arquivo de cabeçalho (integral.h) e o de implementação (integral.c) segue boas práticas de modularidade, permitindo a reutilização e a fácil extensão do código. Cada método de integração oferece diferentes níveis de precisão e desempenho, adequados para uma variedade de funções matemáticas. O ambiente de programação criado é claro, eficiente e escalável.

3. Matrix ADT

3.1 Objetivos

O objetivo principal deste projeto é criar um ambiente de programação em C para manipulação de matrizes através da implementação de operações matemáticas básicas (como soma, subtração, multiplicação) e avançadas (como cálculo de determinante e inversa), utilizando estruturas de dados personalizadas (ADTs) e abordagens modulares com múltiplos arquivos fonte e um Makefile para gerenciar a compilação.

3.2 Introdução Teórica

Matrizes são estruturas matemáticas compostas por elementos dispostos em linhas e colunas, amplamente usadas em diversas áreas da ciência, como álgebra linear, física e computação gráfica. Manipular matrizes computacionalmente envolve implementar operações como:

- **Soma e Subtração de Matrizes:** Operações elementares realizadas componente a componente.
- **Multiplicação de Matrizes:** É mais complexa e envolve somas de produtos dos elementos das linhas e colunas.
- **Determinante de uma Matriz:** Um valor escalar que pode ser calculado de diferentes maneiras, sendo uma métrica importante para identificar se a matriz é invertível.
- **Inversa de uma Matriz:** Utilizada para resolver sistemas de equações lineares, podendo ser calculada quando a matriz possui determinante não nulo.

3.3 Descrição dos Arquivos Fonte

O projeto é composto por três arquivos principais:

- **matrix.h: Arquivo de cabeçalho que contém as declarações da estrutura Matrix e das funções relacionadas à manipulação de matrizes, como inicialização, liberação de memória, e operações como soma, subtração, multiplicação, transposição, e cálculo de determinante e inversa.**
- **matrix.c: Implementação das funções declaradas em matrix.h. Contém o código para criar e destruir matrizes, além de realizar operações matemáticas diversas sobre as matrizes, como soma, subtração, multiplicação por escalar, transposição, cálculo de cofatores, determinante e inversa.**
- **main.c: Arquivo principal que contém a função main() do programa. Aqui, são testadas as funções de manipulação de matrizes implementadas em matrix.c. O arquivo contém exemplos práticos de operações entre duas matrizes e exibe os resultados das operações na tela.**

3.4 Descrição da Estrutura de Diretórios Utilizada

A estrutura do projeto é simples e direta, composta por três arquivos principais em um único diretório.

3.5 Descrição do Arquivo de Compilação (Makefile)

CC: O compilador a ser usado (no caso, gcc).
CFLAGS: As flags de compilação (neste caso, habilita warnings e otimizações de nível 2).
TARGET: O nome do executável a ser gerado.
Regras: Contém regras para compilar os arquivos objeto e linká-los para gerar o executável final.
clean: Remove os arquivos objeto e o executável.

3.6 Descrição das ADTs Criadas (.h e Implementação)

A ADT (Abstract Data Type) criada neste projeto é a Matrix, que representa uma matriz bidimensional de números reais. A estrutura e as funções associadas a esta ADT estão declaradas no arquivo matrix.h e implementadas em matrix.c.

- Estrutura Matrix:
  - int rows: Número de linhas da matriz.
  - int cols: Número de colunas da matriz.
  - double** data: Ponteiro para os dados da matriz, armazenados como uma matriz de valores do tipo double.
- Funções associadas:
  - Matrix* initialize_matrix(int rows, int cols): Inicializa uma matriz de dimensões especificadas.
  - void destroy_matrix(Matrix* matrix): Libera a memória alocada para a matriz.
  - void display_matrix(Matrix* m): Exibe a matriz no console.
  - Matrix* sum_matrices(Matrix* m1, Matrix* m2): Retorna a soma de duas matrizes.
  - Matrix* subtract_matrices(Matrix* m1, Matrix* m2): Retorna a subtração de duas matrizes.
  - Matrix* multiply_matrices(Matrix* m1, Matrix* m2): Retorna o produto de duas matrizes.
  - Matrix* scale_matrix(Matrix* m, double scalar): Multiplica todos os elementos da matriz por um escalar.
  - Matrix* transpose_matrix(Matrix* m): Retorna a transposição de uma matriz.
  - double calculate_determinant(Matrix* m): Calcula o determinante da matriz.
  - Matrix* invert_matrix(Matrix* m): Retorna a inversa da matriz (se possível).

3.7 Conclusões sobre o Ambiente de Programação Gerado

O ambiente de programação desenvolvido oferece uma estrutura organizada para operações matemáticas com matrizes, garantindo modularidade e reusabilidade. As funções implementadas permitem criar, manipular e realizar operações como soma, subtração, multiplicação, inversão e cálculo de determinante de matrizes de maneira eficiente.