Universidade Católica de Brasília

Pró-Reitora Acadêmica
Curso de Engenharia de Software
Trabalho de Disciplina

TRABALHO EM GRUPO ALGORITMOS E PROGRAMAÇÃO

Autor(es): Ana Carolina Lima, Ivo Pinheiro, Ricardo Júnio Santos Marinho, Thiago Vieira Silva, Leticia Lumi Mori Ane e Arthur Vieira de Sousa.

Orientador: Prof. JEFFERSON SALOMÃO RODRIGUES

Brasília - DF
2024

Ana Carolina Lima, Ivo Pinheiro, Ricardo Júnio Santos Marinho, Thiago Vieira Silva, Leticia Lumi Mori Abe e Arthur Vieira de Sousa.

TRABALHO EM GRUPO ALGORITMOS E PROGRAMAÇÃO

Documento apresentado ao Curso de graduação de Bacharelado em Engenharia de Software da Universidade Católica de Brasília, como requisito parcial para obtenção da aprovação nas disciplinas de Algoritmos e Programação Estruturada.

Orientador: Prof. Jefferson Salomão Rodrigues.

Brasília
2024

VARGAS, Ana Carolina.
LUMI, Letícia.
VIEIRA, Thiago.
PINHEIRO, Ivo.
SOUSA, Arthur.
JÚNIO, Ricardo.

Trabalho em grupo Algoritmos e Programação. Ana Carolina Lima, Ivo Pinheiro, Ricardo Júnio Santos Marinho, Thiago Vieira Silva, Leticia Lumi Mori Abe e Arthur Vieira. 28 de setembro de 2024.
Paginação: número.

Universidade Católica de Brasília, 28 de setembro de 2024.
Orientação: Prof. Jefferson Salomão Rodrigues.

Projeto Engenharia Software de autoria de Ana Carolina Lima, Ivo Pinheiro, Ricardo Júnio Santos Marinho, Thiago Vieira Silva, Leticia Lumi Mori Abe e Arthur Vieira de Sousa, intitulado “TRABALHO EM GRUPO ALGORITMOS E PROGRAMAÇÃO”, apresentado como requisito parcial para obtenção da aprovação nas disciplinas de Engenharia Software do Curso de Bacharelado em Engenharia de Software da Universidade Católica de Brasília, defendido e aprovado pela banca examinadora abaixo assinada:

_____

Prof. Jefferson Salomão Rodrigues.
Engenharia Software

Brasília
2024

1 INTRODUÇÃO

No discorrer deste documento teremos como objetivo compartilhar o pensamento obtido pelos integrantes do grupo para solucionar os problemas propostos no trabalho, assim de simplificada demonstrar os caminhos utilizados.

2 METODOLOGIA

Com base nos conhecimentos que nos fora passado em sala, deste modo fizemos o uso de bibliotecas essenciais do C como “stdio“, “limits”, “locale”, “stdlib”, “string”, “func_val”, “math” desta forma pudemos completar todos os desafios solicitados.

3 RELATÓRIO ANÁLISE DO CÓDIGO DA FUNÇÃO DECODIFICADOR

3.1 - Descrição da Função e Comportamento

A função decodificador tem como objetivo converter uma *string* de caracteres hexadecimais em seus equivalentes decimais e depois imprimir os caracteres correspondentes, de acordo com as condições. O código lê até 100 caracteres de entrada, processa-os em pares e utiliza a função *strtol* para a conversão de hexadecimal para decimal.

Caso o valor convertido seja zero, o loop de impressão é interrompido. Além disso, a função *func_val* é usada para determinar se a impressão de um caractere deve ser pulada, com base na lógica que pode variar de acordo com a implementação dessa função externa.

3.2 - Explicação para o Uso de Cada Tipo de Dados

- ***char***: Utilizado para armazenar caracteres de forma individual e em pares do código hexadecimal. A escolha de *char* é adequada, pois lidam-se com dados que representam caracteres.
- ***int***: Usado para o *array* traducao, que armazena a conversão dos pares de caracteres hexadecimais em valores decimais. A escolha de *int* é a mais eficiente, pois os valores convertidos são números inteiros e, quando houver a conversão, os caracteres em hexadecimal partirão para decimal, que são números inteiros.
- ***aux[3]***: Um *array* de *char* é utilizado como variável auxiliar para armazenar os dois caracteres hexadecimais e um caractere nulo que termina a *string*. Isso é necessário para garantir que a conversão funcione corretamente.

3.3 - Considerações sobre a Validação e Controle de Fluxo

A função implementa um controle de fluxo eficaz, interrompendo a execução em caso de valores zero e pulando iterações com base no resultado da função *func_val*. O que assegura que apenas caracteres válidos sejam impressos, otimizando o resultado final e evitando saídas indesejadas.

Essas implementações garantem que o código funcione conforme esperado, com uma estrutura clara e eficiente, aproveitando os tipos de dados adequados para cada operação.

4 RELATÓRIO ANÁLISE DA FUNÇÃO FUNC_VAL

4.1 - Descrição da Função e Comportamento

A função *func_val* calcula um valor baseado em uma equação polinomial de sete graus. Recebe dois parâmetros: um inteiro x e um inteiro b, que influenciam o resultado da equação. A função utiliza coeficientes pré-estabelecidos para calcular um valor que é, então, arredondado e retornado. O uso de *round* garante que o resultado final seja um número inteiro, que é fundamental em situações onde um valor inteiro é necessário para futuro processamento.

4.2 - Explicação para o Uso de Cada Tipo de Dados

- ***int***: Os parâmetros x e b são do tipo *int*, pois a função parece operar com contagens ou índices que são naturalmente inteiros. Isso facilita a integração com outras partes do código que também usam tipos inteiros.
- ***double***: Os coeficientes e o resultado da equação são do tipo *double*, permitindo uma maior precisão em cálculos com números decimais. Isso é necessário para manter a precisão dos resultados, especialmente considerando a presença de potências e a adição de vários termos na equação.

4.3 - Considerações sobre Cálculo e Precisão

A função aplica uma série de operações matemáticas, incluindo multiplicação, potenciação e arredondamento. O uso da biblioteca *math.h* é justificado pela necessidade de operações de potência e arredondamento, o que aumenta a complexidade dos cálculos. A escolha do arredondamento como método de finalização é importante para garantir que o resultado seja um número inteiro, que pode ser utilizado em outras funções do sistema sem causar erros de tipo.

Essas implementações garantem que a função *func_val* opere de forma eficiente e precisa, possibilitando a aplicação dos resultados em cenários onde a integração com outras funções é necessária.

5 RELATÓRIO ANÁLISE DO CÓDIGO PRINCIPAL

5.1 - Descrição do Funcionamento do Programa

O programa principal é responsável por gerenciar a entrada de dados, incluindo a quantidade de mensagens interceptadas e seus valores associados a *b*. Ele começa solicitando ao usuário que insira a quantidade de mensagens e, logo, ele coleta tanto o valor de b quanto a mensagem correspondente para cada uma das entradas. As mensagens são armazenadas em um vetor de *strings*, enquanto os valores de b são armazenados em um vetor de inteiros. Após a coleta, cada mensagem é processada pela função decodificador.

5.2 - Explicação para o Uso de Cada Tipo de Dados

- ***int***: O tipo *int* é usado para o vetor b e para a variável *qtd_mensagens*, porque eles representam contagens e índices que são naturalmente inteiros. O que permite facilidade no gerenciamento e manipulação dos dados ao longo do programa.
- ***char***: O vetor de mensagens é declarado como um *array* de *strings* (com cada *string* tendo até 100 caracteres), próprio para armazenar mensagens interceptadas. O uso de *char* é necessário para lidar com dados textuais, garantindo que os caracteres sejam tratados corretamente.

5.3 - Considerações sobre Coleta de Dados e Controle de Fluxo

O programa utiliza *scanf* para capturar a quantidade de mensagens e os valores de b, seguido por *getchar*() pra limpar o *buffer* de entrada. É importante para evitar problemas com a leitura da *string* da mensagem, uma vez que *scanf* pode deixar um caractere de nova linha que afetaria a entrada subsequente. A função *fgets* é utilizada para garantir que as mensagens sejam lidas corretamente, incluindo espaços.

A estrutura de loops permite a coleta sequencial de dados e a subsequente decodificação, tornando o programa eficiente e fácil de entender. Ao final do processamento, cada mensagem é decodificada com base no seu respectivo valor de b, resultando em uma saída organizada.

6 RELATÓRIO ANÁLISE DO CÓDIGO TABELA DE LIMITES E COMPORTAMENTO DE VARIÁVEIS

1.1. Descrição da Função e Comportamento

A função tem como objetivo principal exibir uma tabela com os limites máximos e mínimos de diferentes tipos de variáveis em C. Além disso, ela demonstra o que ocorre ao atribuir a uma variável um valor fora dos limites suportados, no caso específico de uma variável do tipo *char*.

O código começa definindo a localidade para português, utilizando a função *setlocale*, garantindo que a formatação de caracteres seja adequada. Em seguida, imprime os limites para tipos de dados como *char*, *int*, *long*, entre outros, utilizando as constantes definidas na biblioteca *limits.h*.

Posteriormente, a função demonstra como o tipo *char* lida com o *overflow*. Ao atribuir o valor 300 a uma variável *char*, o valor ultrapassa o limite permitido e é ajustado ao intervalo suportado, conforme explicado no comportamento da tabela ASCII.

1.2. Explicação para o Uso de Cada Tipo de Dados

- ***char***: Utilizado para armazenar um valor numérico dentro do intervalo de -128 a 127. Ao ultrapassar esses limites, ocorre o fenômeno de overflow, onde o valor é ajustado de acordo com o intervalo suportado.
- ***int***: Usado para exibir os limites máximos e mínimos de inteiros, fornecendo precisão adequada para cálculos inteiros de maior magnitude.
- ***unsigned*** int: Esse tipo é utilizado para armazenar apenas valores positivos, ampliando o intervalo disponível para números maiores, uma vez que elimina a possibilidade de valores negativos.
- ***long*** e *long long*: Esses tipos são utilizados para lidar com valores inteiros muito grandes, tanto positivos quanto negativos, fornecendo uma maior faixa de valores.
- ***unsigned*** *long* e *unsigned long long*: Usados para representar números muito grandes sem a possibilidade de valores negativos, ampliando a capacidade de armazenamento.

1.3 Considerações sobre a Validação e Controle de Fluxo

A função demonstra o comportamento do overflow ao atribuir o valor 300 a uma variável do tipo char, que não é capaz de armazenar esse valor diretamente. Isso resulta em um ajuste do valor, conforme ele "dá a volta" no intervalo suportado pelo tipo char. Este comportamento é utilizado para ilustrar como os tipos de dados lidam com valores fora dos limites.

Além disso, a função imprime uma tabela bem estruturada com os limites das variáveis, o que ajuda a compreender os valores máximos e mínimos que cada tipo de dado pode armazenar. Isso é importante para garantir a escolha adequada dos tipos de dados em diferentes situações de programação, evitando erros como overflow.

7 RELATÓRIO ANÁLISE DO CÓDIGO PARA CÁLCULO DE NOTAS

1.1 - Descrição da Função e Comportamento

A função tem como objetivo principal calcular a nota final de um aluno com base nas avaliações realizadas e suas respectivas notas. O código solicita ao usuário que insira as notas n1, n2 e a nota do PPD (Participação Discente), validando se elas estão dentro dos limites permitidos (de 0 a 4.5 para n1 e n2, e de 0 a 1 para ppd).

Além disso, o código verifica se o aluno realizou o Exame Unificado (EU) e/ou a N3 (terceira prova). Se o aluno realizou a N3, o código substitui a menor nota entre n1 e n2 pela nota n3, melhorando a média do aluno. Após todas as validações e ajustes, a nota final é calculada. Caso seja inferior a 9.9, o código adiciona a nota do Exame Unificado à nota final.

O comportamento desta função é estruturado de forma clara, com um fluxo lógico semelhante ao de uma função que decodifica e processa dados passo a passo, garantindo a validação correta das entradas e o processamento das notas de acordo com as condições estabelecidas.

1.2 - Explicação para o Uso de Cada Tipo de Dados

- ***float***: Utilizado para armazenar as notas (n1, n2, n3, ppd, nEU, nFinal e ap). A escolha de ***float*** é adequada, pois as notas podem conter valores fracionados, sendo necessário manipular números com precisão decimal.
- ***int***: Utilizado para variáveis que representam escolhas binárias, como fn3, que indica se o aluno realizou ou não a N3. Esse tipo de dado é eficaz para representar respostas simples de 0 ou 1, economizando memória e facilitando o controle de fluxo.

Essas escolhas garantem que os tipos de dados utilizados sejam apropriados para as operações executadas, assim como no uso de char e ***int*** na função de decodificação, onde cada tipo de dado desempenha um papel específico e necessário no processamento.

8 RESULTADOS:

A função foi implementada com sucesso, cumprindo seu objetivo de calcular a nota final do aluno com base nas diferentes avaliações, como N1, N2, PPD e Exame Unificado (quando aplicável). O código realiza a validação das entradas, assegurando que os valores inseridos estejam dentro dos intervalos permitidos e processa as notas corretamente.

1. **Validação de Entradas:**
   A função garante que as notas n1 e n2 estejam no intervalo de 0 a 4.5, enquanto ppd e a nota do Exame Unificado são validadas entre 0 e 1. Isso previne erros na inserção de dados e garante que o cálculo da nota final seja feito com valores consistentes.

2. **Substituição de Notas:**
   Quando o aluno realiza a N3, o código substitui a menor nota entre n1 e n2 pela nota n3, aumentando as chances de melhorar a média final do aluno. Esta lógica assegura que o aluno seja beneficiado por sua melhor nota, otimizando o cálculo da nota final.

3. **Cálculo da Nota Final:**
   O cálculo da nota final é realizado somando n1, n2 e ppd. Caso o valor seja inferior a 9.9, a nota do Exame Unificado é adicionada, permitindo que o aluno melhore sua pontuação final. Se a nota final ultrapassar 10, ela é ajustada para o valor máximo de 10.

4. **Resultado Final:**
   A função determina se o aluno foi aprovado ou reprovado, com base na nota final ajustada. Se a nota for igual ou superior a 6, o aluno é considerado aprovado; caso contrário, é reprovado.

Esses resultados mostram que o código funciona conforme o esperado, garantindo a correta validação, processamento e cálculo da nota final, similar às funções analisadas anteriormente, onde o controle de fluxo e a validação de dados foram fundamentais para o bom funcionamento do programa

9 CONCLUSÃO:

A função desenvolvida faz seu trabalho com bastante eficiência. Dadas as avaliações, com regras pré-definidas, calcula a nota final dos alunos. Além disso, há uso apropriado de tipos de dados: float para armazenar notas e int para variáveis de controle binárias para aumentar a precisão e a eficiência no processamento de dados. Além disso, a validação de entrada não deve ser ignorada para evitar problemas. A validação garantirá que todas as notas inseridas estejam dentro da faixa permitida.

Esse código também fornece a lógica adequada para substituir a menor nota por n3, quando aplicável, e adicionar a nota do Exame Unificado quando a soma das notas for insuficiente para aprovação do aluno. O controle de fluxo é robusto e evitam-se inconsistências para ajustar a nota final e não ultrapassar o valor máximo de 10.

Tal como acontece com todas as análises sobre funções de validação e controle anteriores a esta, este exemplo novamente representa o uso eficiente de condicionais e loops para garantir a integridade de sua saída. A lógica realizada nesta implementação garante que o cálculo da nota seja feito de forma totalmente justa e correta e chegue claramente à conclusão com o desempenho do aluno.

10 FONTE:

jefferson-rodrigues-ucb/codigos-aula-ape - GitHub

11 APÊNDICE:

https://github.com/He1ce/Trabalho-N1-APE