Algoritmos e Programação Estruturada
Trabalho em Grupo N1

v:1.0.2

1 Instruções Iniciais

Redija um artigo acadêmico contendo os seguintes itens:

- **Título**: Deve ser claro e conciso, refletindo o conteúdo do artigo.
  - **Autores**: Os integrantes do grupo que realizaram o trabalho.
- **Resumo**: Um breve resumo do estudo, incluindo objetivos, métodos, resultados e conclusões.
- **Introdução**: Apresenta o tema, a relevância do estudo e os objetivos da pesquisa.
- **Metodologia**: Descrição detalhada dos métodos e procedimentos utilizados na pesquisa.
- **Resultados**: Apresentação dos dados coletados e das análises realizadas.
  - **Discussão**: Interpretação dos resultados e suas implicações.
- **Conclusão**: Resumo dos principais achados.
- **Referências**: Lista de todas as fontes citadas no artigo.
- **Apêndice**: Adicione o link do repositório git onde está hospedado todo o código utilizado no trabalho.
  - Caso seja um repositório privado, lembrem de adicionar meu e-mail: jefferson.rodrigues@p.ucb.br.

2 Códigos-fonte

Crie um repositório git (Github ou Gitlab).
Todos os código produzidos devem estar em arquivos *.c*, *.h* ou *Makefile*.
Algumas dicas para códigos bem feitos:

- **Nomes Claros e Significativos**: Use nomes descritivos para variáveis, funções e classes. Por exemplo, calcularMedia é mais claro do que calc.
- **Simplicidade e Clareza**: Escreva funções pequenas que realizam uma única tarefa. Isso facilita a leitura e a manutenção do código.
- **Comentários Úteis**: Comente apenas o necessário para explicar partes complexas do código. Evite comentários óbvios que não agregam valor.
- **Indentação Consistente**: Mantenha uma indentação consistente para melhorar a legibilidade. Isso ajuda a visualizar a estrutura do código facilmente.
- **Evite Repetição**: Siga o princípio DRY (Don't Repeat Yourself). Reutilize código sempre que possível para evitar duplicação.
- **Seja honesto**: Você não aprende nada copiando código de terceiros (isso inclui IAs) nem pedindo a uma pessoa externa ao grupo que faça o trabalho por você. Se a cópia for detectada, a nota de vocês será zerada e as devidas providências serão tomadas.

3 Desafio

O ano é 1944 e o Brasil está na Segunda Guerra Mundial. Você está em uma equipe dos Aliados encarregada de decodificar mensagens do inimigo. O único dispositivo que sua equipe possui é uma antena que consegue captar ondas de rádio. Sua equipe recebe vários sinais na antena mas não consegue entender nada do que se passa.

Para sua sorte, a equipe de inteligência conseguiu relacionar o código recebido conforme tabela *ascii*. Contudo, percebeu-se que, além da cifra, alguns caracteres não eram válidos e precisavam ser ignorados. Uma pessoa da equipe de inteligência conseguiu relacionar a posição desses valores com a seguinte função:

$$f(x, b) = a_0 + (a_1 + b)x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$$

Em que,

- $a_0 = 186,752$
- $a_1 = -148,235$
- $a_2 = 34,5049$
- $a_3 = -3,5091$
- $a_4 = 0,183166$
- $a_5 = -0,00513554$
- $a_6 = 0,0000735464$
- $a_7 = -4,22038 \times 10^{-7}$
- b é uma variável que será dada pelo agente da inteligência a cada interceptação.
- x é a posição atual do caractere, começando por 1.

Sua missão é criar um algoritmo que decifre as mensagens intercepadas.

A mensagem está codificada em hexadecimal, em que cada 2 caracteres da mensagem representa um caractere da mensagem, ou seja, cada mensagem pode ter até 50 caracteres.

O texto é encerrado se encontrado o caractere nulo, representado pelo valor 00, o valor 00 encerrará o texto independente do resultado da função.

Imprima a mensagem decodificada.

Utilize o cmd do Windows (Prompt de Comando), deve ser um terminal que utiliza a codificação `CP437` (também conhecida como OEM 437 ou DOS Latin US).

3.1 Exemplo de caso de teste

3.1.1 Entrada

O número de mensagens interceptadas ($[1, 10000]$).

Cada mensagem interceptada é dada por um número que represente a variável b ($[-100, 100]$), em seguida, a mensagem codificada em hexadecimal, 100 caracteres, cada 2 caracteres representa um número.

3.1.2 Saída

A mensagem decodificada.

3.1.3 Exemplos:

Entrada	Saída
1 0 566F6388732073C66F2076656_ E6365646F867265732C20766F_ 63C3887320636FBE6E7365677_ 5656D2E002DC6C921B7B87FCF	Vocês são vencedores, vocês conseguem.
1 3 5465636E6F336C6f676961206_ 46120496E666f726D6187C66F_ 2E00333333333333333333333_ 33333333333333333333333333	Tecnologia da Informação.
2 0 566F6388732073C66F2076656_ E6365646F867265732C00566F_ 6388732073C66F2076656E636_ 5646F867265732C00332C2C2C 3 566F638873C320636F6E73656_ 775656D2E002DC6C921B7B87F_ CF566F638873C320636F6E736_ 56775656D2E002DC6C921B7B8	Vocês são vencedores, Vocês conseguem.

Observação: a string foi separada por _ porque não coube na folha. Nos testes, a string virá em sequência, em uma única linha.

3.2 Atividades

3.2.1 Análise dos limites suportados pelos tipos de arquivos

Uma causa importante de erros em programação em C é ultrapassar os limites numéricos estabelecidos para cada tipo de variável. O objetivo desse exercício é mostrar como e porque ocorrem esses erros. Os limites numéricos que cada tipo inteiro estão definidos na biblioteca da linguagem `LIMITS.H`, na forma de constantes. Escreva um programa que use essa biblioteca e imprima os limites mínimos e máximos que podem ser assumidos por variáveis dos tipos `char`, `int`, `short int`, `unsigned int`, `long int`, `unsigned long int`, `long long int`, `unsigned long long int`. Apresente os resultados em uma tabela. Faça um pequeno programa, usando uma variável à qual é atribuída um valor maior do que seu tipo permite, para demonstrar o problema que pode ocorrer.

Explique o que acontece se você utilizar um valor fora dos limites dos tipos de variáveis.

Em seu relatório, não esqueça de mencionar o porquê do uso de cada tipo para as variáveis utilizadas em sua implementação.

3.2.2 Função que invalida alguns caracteres

Implemente uma função que realize o cálculo da função proposta conforme *lib*:

```
1 #ifndef REMOVE_H
2 #define REMOVE_H
3
4 int func_val(int x, int b);
```

```
5
6   #endif        /* REMOVE_H */
```

Implemente a função func_val, utilize a função `round` da biblioteca *MATH.H* para arredondar o valor do resultado da função.

Se a função retornar 0, o caractere deve ser ignorado.

3.2.3 Função Principal

A função principal deverá importar, além das bibliotecas necessárias, a função implementada em 3.2.2.
Faça a leitura do número de casos de testes.
Então, para quantidade de casos de testes, faça a leitura da variável b e da `string` contendo a mensagem cifrada.
Converta a cada 2 caracteres para o valor decimal e imprima a mensagem.
Apenas a mensagem deve ser impressa, qualquer impressão além causará erros na análise automática.
Lembre-se de justificar cada escolha no relatório.

4 Desafio Extra

Esse desafio é extra, não realizar ele não impede de conseguir todos os pontos do trabalho.

Construa um algoritmo em C que implemente o sistema de avaliação da UCB.
O programa irá receber as três notas principais (N1, N2 e PPD), valide se:

- N1 = $[0; 4, 5]$
- N2 = $[0; 4, 5]$
- PPD = $[0; 1]$

Caso algum não esteja com valor válido, retorne código de erro 3.
Em seguida, receba uma *flag* que determinará se o estudante realizou o Exame Unificado (EU), 0 para não e 1 para sim.
Caso tenha feito, receba a nota do exame unificado, lembre de validar se EU = $[0; 1]$.
Também receba uma *flag* que defina se o estudante realizou a N3, 0 para não e 1 para sim.
Caso tenha realizado, leia a N3 e também verifique se ela está no intervalo permitido, N3 = $[0; 4, 5]$ e substitua a menor nota na composição da nota final.
Lembre-se que a nota final deve estar no intervalo $[0; 10]$.
Apresente a nota final e se o estudante foi aprovado ou reprovado.

4.1 Exemplo

Entrada	Saída
4.1 2.2 0.78 1 0.6 0	 7,68 aprovado