

**Pró-Reitoria Acadêmica
Curso de Engenharia de Software**

**TRABALHO DE DESCRIPTOGRAFIA
NA LINGUAGEM C**

**Autores: Rubens de Oliveira Cristiano,
Vinícius Ribeiro Peixoto**

Orientador: Jefferson Salomão Rodrigues

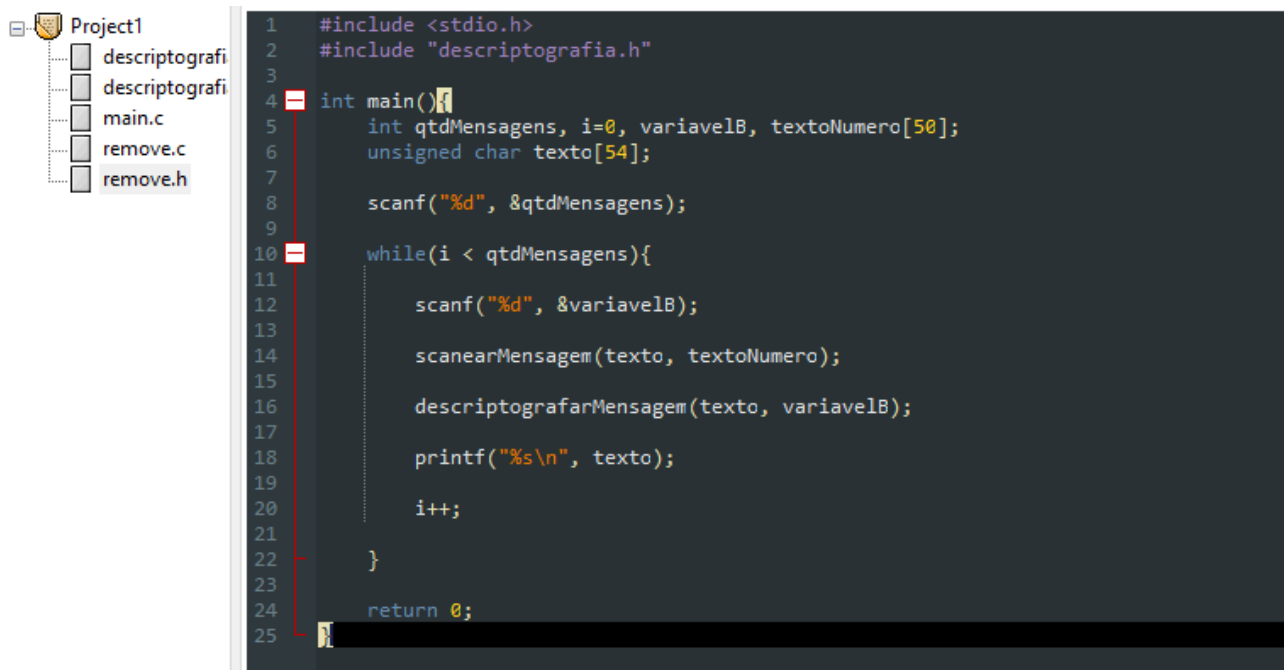
RESUMO:

Neste artigo, vamos explorar o código que desenvolvemos para um trabalho que tinha como objetivo fortalecer alguns conceitos de C. O sistema a ser feito deveria ser capaz de descriptografar uma mensagem em hexadecimal. Primeiro o sistema receberia uma quantidade de mensagens que poderia ser de 1 a 10000, e depois iria receber uma variável B seguida da mensagem. Cada nova mensagem teria uma variável B própria. Por fim, o sistema iria pegar todas essas mensagens e hexadecimal e mostrar o texto que elas representam após remover os caracteres indesejados que forem encontrados. E os testes que realizamos mostram que o sistema cumpre com seu propósito, pois foi capaz de realizar a descriptografia das mensagens.

INTRODUÇÃO:

Neste trabalho, utilizamos a linguagem C e as bibliotecas <stdio.h>, <math.h> e outras criadas durante o projeto que são “remove.h” e “descriptografia.h”, e este estudo se mostrou relevante para aperfeiçoar os conhecimentos sobre loops, variáveis, arrays, estruturas de condicionamento, e lógica de programação. E o objetivo final do sistema era pegar as mensagens criptografadas e remover os caracteres desnecessários e imprimir a mensagem final completamente descriptografada. Este estudo foi muito importante para desenvolver e fixar os conceitos de C que aprendemos ao longo do bimestre e para também melhorar a nossa habilidade de resolução de problemas.

METODOLOGIA:



```
1  #include <stdio.h>
2  #include "descriptografia.h"
3
4  int main(){
5      int qtdMensagens, i=0, variavelB, textoNumero[50];
6      unsigned char texto[54];
7
8      scanf("%d", &qtdMensagens);
9
10     while(i < qtdMensagens){
11         scanf("%d", &variavelB);
12
13         scanearMensagem(texto, textoNumero);
14
15         descriptografarMensagem(texto, variavelB);
16
17         printf("%s\n", texto);
18
19         i++;
20     }
21
22     return 0;
23
24
25 }
```

Figura 1 – Trecho do código em C do arquivo main.c, responsável pelo escopo principal do projeto chamando as funções que vão ser utilizadas.

No código que construímos a gente o dividiu em 5 arquivos para modularizar e tornar sua manutenção mais simples

Linha 1: Nesta linha implementamos a biblioteca <stdio.h> para poder utilizar algumas das suas funções dentro do nosso projeto.

Linha 2: Implementamos a biblioteca que criamos chamada “descriptografia.h” onde colocamos algumas funções essenciais para o funcionamento do código.

Linha 4: começamos a função int main() que é a função principal do código, é onde o código começa, onde desenvolvemos todo o corpo do código.

Linha 5: declaramos as variáveis qtdMensagens, i, variavelB. E o vetor textoNumero.

qtdMensagens: É uma variável usada para pegar a primeira entrada do usuário que no caso seria a quantidade de mensagens que o usuário deseja descriptografar, sempre entre 1 e 10000.

i: É uma variável usada para controlar o loop while, que lê as mensagens criptografadas, as descriptografa e imprime.

variavelB: É uma variável usada para receber a entrada da variável B que é mandada de entrada toda vez que uma nova mensagem é enviada, sempre entre -100 e 100.

textoNumero: Um vetor do tipo int criado para armazenar os valores hexadecimais lidos e convertidos para decimal. Esses valores são transferidos para o array de char “texto”, que é utilizado para a organização e exibição final da mensagem descriptografada.

Linha 6: Declaramos um vetor unsigned char texto[54], que é onde vamos armazenar a mensagem descriptografada, a mensagem é enviada em 100 dígitos e a cada 2 dígitos é um caractere, então este array só precisa ter 51 de espaço contando com o \0, porém nós colocamos 54 pois estávamos tentando evitar um erro que estava acontecendo em questões de sobreposição de memória, pois o endereço da variavelB e do texto estavam muito próximos um do outro e quando colocávamos valores dentro do texto acabava sobrepondo a variavelB e por isso colocamos mais 3 espaços para evitar problemas na memória, e nós colocamos ele como unsigned para que ele possa pegar caracteres da tabela de ASCII estendida, permitindo valores de 0 a 255.

Linha 8: Utilizamos scanf para capturar a quantidade de mensagens, que deve ser um número inteiro entre 1 e 10.000.

Linha 10: Aqui criamos o loop com a condição de enquanto o i for menor que a qtdNumeros, este loop serve para fazer com que o sistema leia exatamente a quantidade de mensagens que o usuário havia dito que mandaria, sempre lendo uma variavelB e uma mensagem de 100 dígitos.

Linha 12: Esse scanf lê a variavelB fornecida pelo usuário e está dentro do laço, pois, a cada nova mensagem, uma nova variavelB deve ser fornecida, e esta variável vai ser utilizada na função func_val, que será vista mais para frente, que vai servir para descartar os caracteres que não deviam estar no texto descriptografado.

Linha 14: Esta linha chama a função scanearMensagem e manda de parâmetro o vetor unsigned char “texto”, e o vetor int “textoNumero”, e esta função que será vista mais para frente serve para ir scanear 2 hexadecimal por vez e ir transformando em decimal que também é um ASCII para um caractere e ir armazenando estes valores decimais primeiro no array “textoNumero” e depois no array “texto”.

Linha 16: Aqui nos chamamos outra função que seria a `descriptografarMensagem` e de parâmetros mandamos o vetor “texto” e a variável `B`, e esta função que também será vista mais para a frente, serve para pegar o texto já scaneado e transformado de hexadecimal em decimal, e vai descartando os caracteres quando a `func_val` retornar 0, e descartar todos os outros caracteres que aparecem depois do 00.

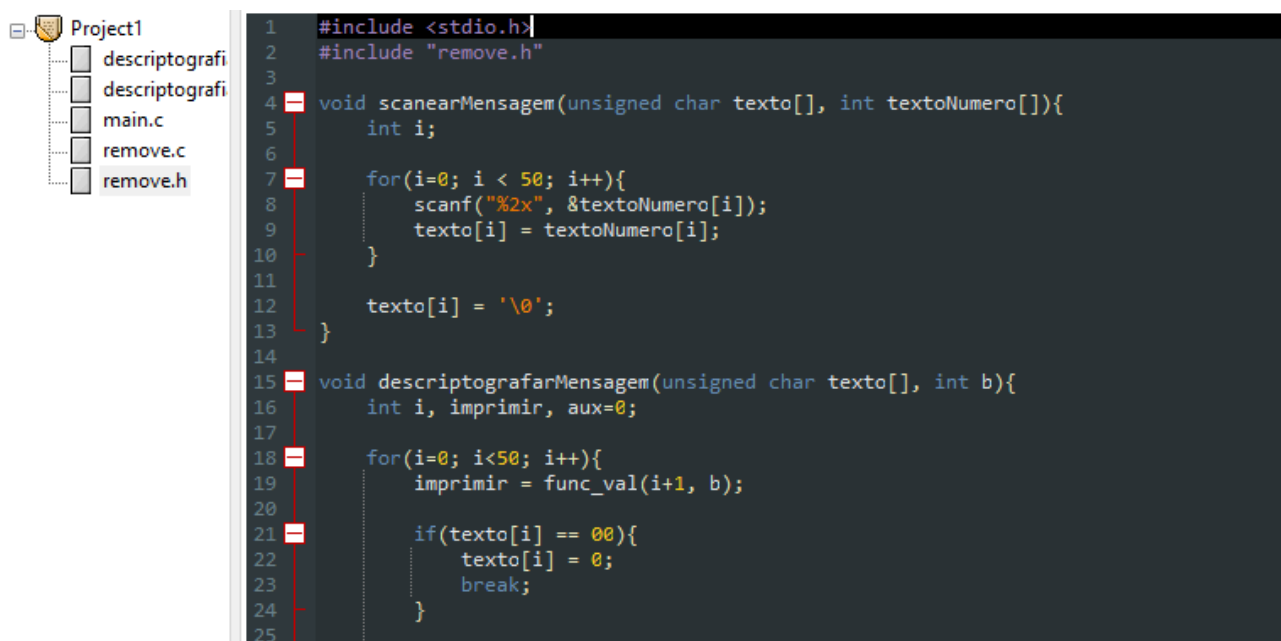
Linha 18: Nesta linha colocamos um `printf` para ir imprimindo cada texto descriptografado e organizado, a cada mensagem que for sendo descriptografada o sistema vai ir imprimindo.

Linha 20: aqui colocamos um incremento no `i` para que o loop do `while` possa alcançar o fim quando o `i` for igual a `qtdMensagens`, fazendo com que o loop leia com exatidão somente as quantidade de mensagens que o sistema havia recebido como primeira entrada.

Linha 22: Fechamento do loop `while`.

Linha 24: Colocamos um `return 0` para finalizar o programa e também para verificar se o programa foi encerrado corretamente.

Linha 25: Fechamento da função `int main`.



```
1  #include <stdio.h>
2  #include "remove.h"
3
4  void scanearMensagem(unsigned char texto[], int textoNumero[]){
5      int i;
6
7      for(i=0; i < 50; i++){
8          scanf("%2x", &textoNumero[i]);
9          texto[i] = textoNumero[i];
10     }
11
12     texto[i] = '\0';
13 }
14
15 void descriptografarMensagem(unsigned char texto[], int b){
16     int i, imprimir, aux=0;
17
18     for(i=0; i<50; i++){
19         imprimir = func_val(i+1, b);
20
21         if(texto[i] == 00){
22             texto[i] = 0;
23             break;
24         }
25 }
```

Figura 2 – Trecho do código em C do arquivo `descriptografia.c`, mostrando a primeira função “`scanearMensagem`” e o começo da função “`descriptografarMensagem`”.

Este é o arquivo `.c` que criamos onde estão as funções que utilizamos dentro do arquivo principal.

Linha 1: Nesta linha nós implementamos a biblioteca `<stdio.h>` para poder usar algumas das suas funções e funcionalidades dentro do nosso código.

Linha 2: Nesta linha nós implementamos a biblioteca “remove.h” que foi uma biblioteca que criamos com a função que foi passada na atividade que seria a “func_val” que veremos mais para a frente.

Linha 4: Aqui começamos a criar a primeira função que seria a “scanearMensagem” que de parâmetro ela pede um array de unsigned char e nomeia ele como “texto”, e pede array de int e nomeia ele como “textoNumero”, e o principal intuito desta função é de pegar toda a mensagem criptografada em hexadecimal, transformar cada 2 dígitos do hexadecimal em decimal que também é um caractere em ASCII e guardar no array textoNumero que depois vai guardar no vetor “texto”. Essa função é do tipo void pois ela não vai retornar nada, vai apenas armazenar valores dentro dos vetores

Linha 5: Declaramos a variável inteira “i” que será utilizada para controle do loop.

Linha 7: Declaramos o loop que vamos utilizar para scanear toda a mensagem, como a mensagem sempre será de 100 dígitos o loop vai ser repetido apenas 50 vezes já que a cada interação ele lê 2 dígitos e já transforma para decimal.

Linha 8: Nesta linha nós colocamos o %2x para escanear 2 dígitos de hexadecimal e transformar para decimal. E este decimal também é um caractere em ASCII, e este decimal vai ser armazenado no array de int “textoNumero”, pois em alguns compiladores não é aceito utilizar o %2x diretamente em um array de char, e por isso primeiro o valor é scaneado e armazenado no “textoNumero” e, em seguida, é posto no “texto”, e ambos os vetores tem seus índices controlados por “i” que é a variável de controle do loop que vai de 0 a 49, permitindo pegar os 50 caracteres.

Linha 9: Aqui colocamos o vetor “texto” recebendo o valor decimal do vetor “textoNumero” que futuramente será transformado na mensagem final descriptografada.

Linha 10: Fechamento do loop for.

Linha 12: Nesta linha nós pegamos o array texto e pegamos a posição do índice “i” que neste momento do código tem o valor de 50, e nesta posição colocamos o “\0” que serve para mostrar que a string vai só até este ponto.

Linha 13: Fechamento da função “scanearMensagem”.

Linha 15: Aqui nós começamos a função “descriptografarMensagem”, que é uma função que pede dois parâmetros, um array de unsigned char que será chamada de “texto”, e uma variável do tipo int que será chamada de “b”. Esta função tem como objetivo pegar aquele array de texto já com os valores decimais dentro de cada índice que no caso são os caracteres, e remover os caracteres que não devem estar no resultado utilizando a função func_val, e também pela regra de ignorar todos os caracteres que aparecem na frente do hexadecimal 00.

Linha 16: aqui nós declaramos três variáveis do tipo int: i , imprimir , aux.
i : vai ser uma variável usada para o controle do loop for.

imprimir : vai ser uma variável que vai receber o valor de retorno da função func_val e assim vai determinar quais caracteres deverão ou não ser impressos na descriptografia final. Quando func_val retornar 0 o caractere deve ser ignorado, caso contrário o caractere deve ser mantido.

aux : uma variável auxiliar que vai ser utilizada na logica de remover os caracteres quando necessário, pois ela é usada dentro do loop for e vai sendo incrementada a cada repetição do loop e

por isso vai ir tendo o mesmo valor de *i*, porém quando um caractere tiver que ser removido ela não será incrementada e isso vai servir de base para toda a logica desta função que veremos mais para a frente.

Linha 18: Implementação do laço “for” com a variável “*i*” como contador, começando em 0 e incrementando-o até que seja menor que 50, que é o tamanho máximo para a mensagemcriptografada. O laço terá a funcionalidade de percorrer toda a mensagemcriptografada e à partir da função “func_val” eliminar os caracteres indesejados e aqueles após o hexadecimal “00”.

Linha 19: Atribuindo o valor retornado pela função “func_val” na variável “imprimir”, que por sua vez determinará quais caracteres devem ser impressos, se retornar 0 é porque o caractere deve ser ignorado caso retorne um valor diferente de 0 aquele caractere deve ser impresso.

Linha 21: Criação de um “if” que verifica se o elemento do vetor “texto” do índice do atual laço (“*i*”) do “for” é igual ao hexadecimal “00”. A função desse “if” é verificar se a string contem o valor 00 que seria o final da string estipulada no exercício caso aparecesse na mensagem, caso sim, é atribuído “\0” ao atual índice do vetor texto (“*i*”), para encerrar a string, e por fim é dado um “break” para encerrar a execução do laço “for”, pois a descriptografia está completa.

Linha 22: Atribuindo o “\0” ao vetor “texto” no atual índice, para encerrar a string.

Linha 23: Introdução do break para encerrar o laço “for”, devido a string já ter chegado ao final.

Linha 26: Atribuindo o valor do vetor “texto” no índice da variável “*i*”, para o índice auxiliar, que ate o momento é o mesmo do contador “*i*”.

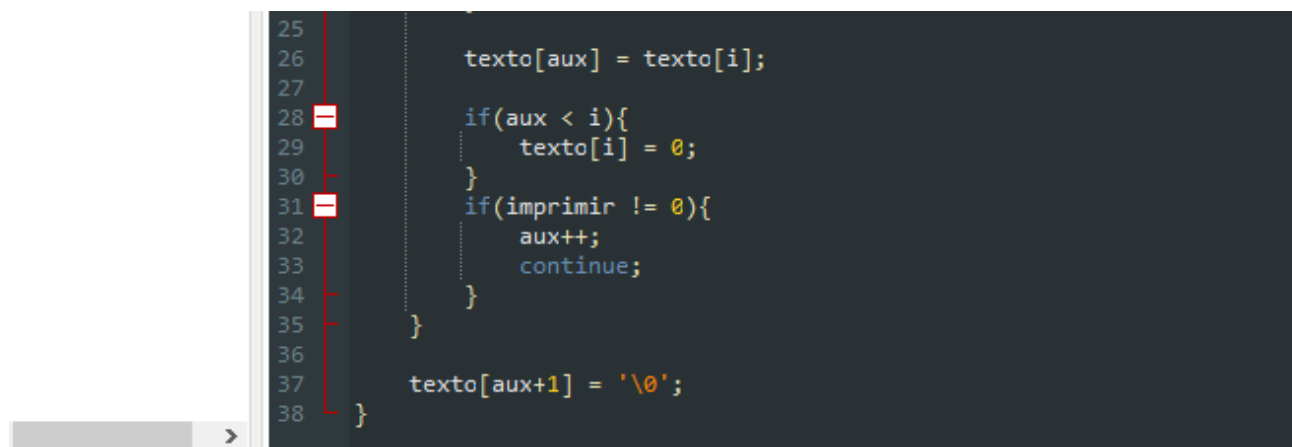
A imagem mostra um trecho de código em C em um editor de texto com fundo escuro. O código está entre as linhas 25 e 38. A linha 25 é vazia. A linha 26 contém 'texto[aux] = texto[i];'. A linha 27 é vazia. A linha 28 contém 'if(aux < i){'. A linha 29 contém ' texto[i] = 0;'. A linha 30 contém '}'. A linha 31 contém 'if(imprimir != 0){'. A linha 32 contém ' aux++;'. A linha 33 contém ' continue;'. A linha 34 contém '}'. A linha 35 contém '}'. A linha 36 é vazia. A linha 37 contém 'texto[aux+1] = '\0';'. A linha 38 contém '}'. Há uma barra vermelha vertical à esquerda das linhas 28 e 31, com um sinal de igual vermelho em cada uma delas. Há também uma barra vermelha vertical à esquerda da linha 35.

Figura 3- Trecho do código em C do arquivo “descriptografia.c”, continuação da função “descriptografarMensagem”.

Linha 28: Criação de um “if” que verifica se a variável “aux” é menor que o contador do laço (“*i*”). O objetivo dessa verificação é garantir que o caractere não seja duplicado, porque, ao se atribuir o valor do vetor texto no índice “*i*” ao mesmo vetor de índice “aux” as duas ficam com o mesmo valor. Se o aux estiver menor do que *i*, que é a condição imposta, significa que um caractere que não pertence a string foi encontrado, e, portanto, como houve a atribuição de “*i*” em “aux” previamente, o índice posterior “*i*”, e o agora anterior, já que não foi incrementado, “aux”, têm o mesmo valor, mas índices diferentes, portanto, dentro dessa condição o valor da variável texto em índice “*i*” passa a ser nulo para evitar repetição.

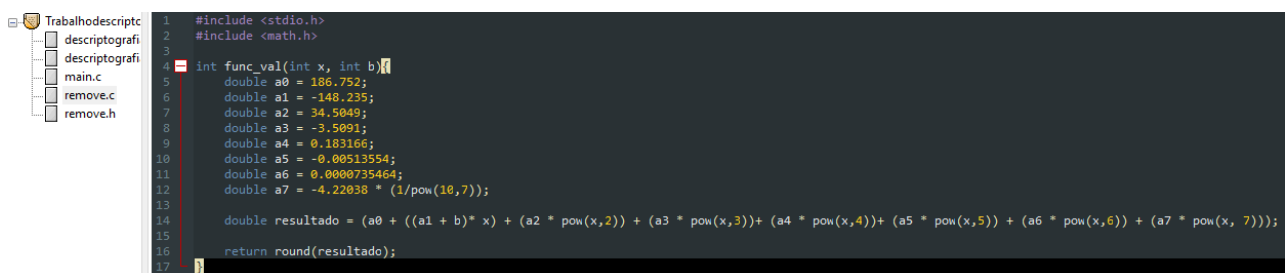
Linha 29: Zerando o valor do vetor “texto” em índice “i”, para evitar que o caractere desse índice e do índice anterior (“aux”) seja o mesmo, e acabem ficando duplicados.

Linha 31: Criação de um “if” que verifica se o valor de retorno da função func_val é 0, sendo utilizado a variável “imprimir” porque ela que contém o valor de retorno da função. O objetivo dessa função é verificar se a função é verdadeira, ou seja, se o caractere da posição que ela entregou, existe na string. A verificação é feita por meio da condição de que “imprimir” seja diferente 0. Caso a condição seja verdadeira a variável “aux” é incrementada, indicando que o valor dela é de “i” são os mesmos, e, portanto, todos os caracteres até o momento do vetor “texto” são realmente da mensagem, caso seja 0 o if vai ser pulado e “aux” vai acabar se tornando menor que “i”, e vai entrar na logica anterior onde vai remover este caractere que deve ser tirado.

Linha 32: A variável “aux” é incrementada, pois até o momento todos os caracteres pertencem a string, e, portanto, não há necessidade de mover caracteres.

Linha 33: Introdução do “continue” para começar o próximo laço sem verificar as linhas seguintes.

Linha 37: Como nem toda mensagem terminará com o hexadecimal “00”, após o encerramento do loop “for”, é atribuído manualmente o “\0” na última posição da string “texto”.

A screenshot of a code editor with a dark theme. On the left, a file explorer shows a project named 'Trabalhodescript' with subfolders 'descriptografi' and 'main.c', and files 'remove.c' and 'remove.h'. The main editor displays the code for 'remove.h'. The code includes standard headers, defines a function 'func_val' that takes an integer 'x' and a boolean 'b', and calculates a polynomial result using various coefficients and powers of 'x'. The result is rounded and returned. Line numbers 1 through 17 are visible on the left margin of the code block.

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int func_val(int x, int b)
5 {
6     double a0 = 186.752;
7     double a1 = -148.235;
8     double a2 = 34.5049;
9     double a3 = -3.5091;
10    double a4 = 0.183166;
11    double a5 = -0.00513554;
12    double a6 = 0.000735464;
13    double a7 = -4.22038 * (1/pow(10,7));
14
15    double resultado = (a0 + ((a1 + b)* x) + (a2 * pow(x,2)) + (a3 * pow(x,3)) + (a4 * pow(x,4)) + (a5 * pow(x,5)) + (a6 * pow(x,6)) + (a7 * pow(x, 7)));
16
17    return round(resultado);
18 }
```

Figura 4 – Trecho do código em C do arquivo remove.h, mostrando a função “func_val”, que foi entregue dentro do trabalho.

Este seria o arquivo da func_val que foi a função dada dentro do trabalho que iria receber a posição do caractere e o valor da variável B, e com isso ele iria devolver se a posição daquele caractere deve ser incluída no texto descriptografado ou não, se a função devolver 0 o caractere deve ser removido e se devolver diferente de 0 o caractere deve ser incluso. Esta função é fundamental para remover alguns caracteres e deixar as frases finais corretas.

Linha 1: Inclusão da biblioteca stdio.h a qual disponibiliza funções de entrada e saída de dados

Linha 2: Inclusão da biblioteca math.h a qual disponibiliza funções relacionadas a operações matemáticas

Linha 6 a 13: Atribuindo os devidos valores as variáveis, do tipo double, pertencentes a função dada pelo professor.

Linha 15: Atribuindo a variável “resultado”, do tipo double, toda a função fornecida pelo professor para a remoção dos caracteres desnecessários da mensagem.

Linha 17: Retorna a variável “resultado” arredondada pela função “round” da biblioteca math.h.

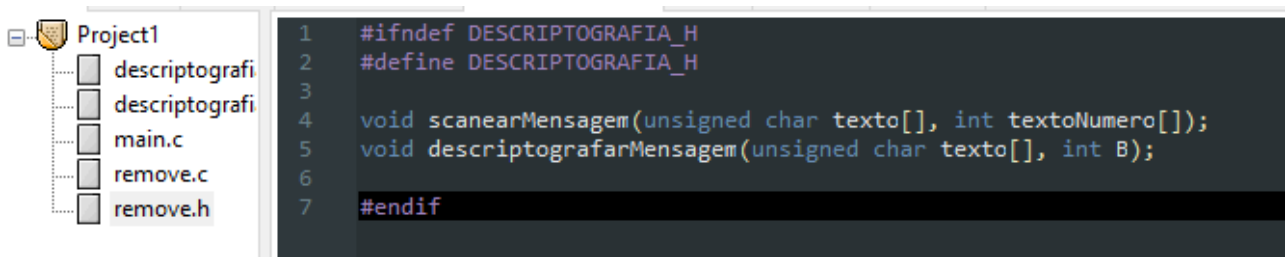


Figura 5 – Trecho do código em C do arquivo “descriptografia.h” com as assinaturas das funções do arquivo “descriptografia.c”.

Aqui nós criamos o arquivo .h da descriptografia.

Linha 1: colocamos o #ifndef para verificar se a descriptografia.h já havia sido definida.

Linha 2 : Nesta Linha colocamos o #define para definir a descriptografia.h caso ela ainda não tenha sido definida.

Linha 4: colocamos a assinatura da função “scanearMensagem” presente dentro da descriptografia.c.

Linha 5: colocamos a assinatura da função “descriptografarMensagem” que também é uma função presente na descriptografia.c.

Linha 7: e por fim, colocamos o #endif para sinalizar o final da leitura do .h.

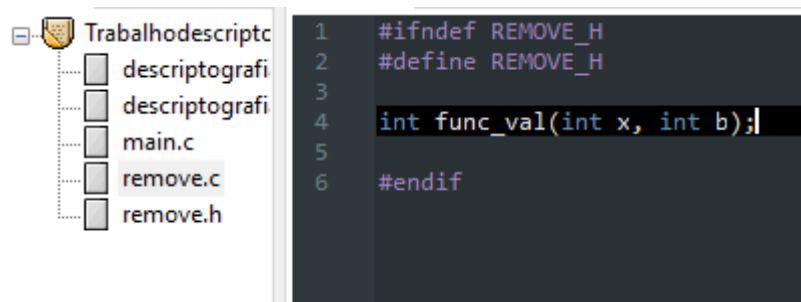


Figura 6 – Trecho do código em C do arquivo “remove.h” com a assinatura da função “func_val” que está dentro do arquivo “remove.c”.

Linha 1: Aqui nós colocamos o #ifndef para ver se o remove_h já havia sido definido.

Linha 2: Nesta parte colocar o #define para definir o remove.h caso ele ainda não tenha sido

Linha 4: Aqui nós colocarmos a assinatura da função func_val que é uma função que está no remove.c

Linha 6: Por fim colocamos o #endif para mostrar o fim da leitura do arquivo .h.

Resultados:

```
C:\Users\User\Desktop\todos os testes de C:\descriptografia\Trabalhodescriptografia.exe
1
0
566F6388732073C66F2076656E6365646F867265732C20766F63C3887320636FBE6E73656775656D2E002DC6C921B7B87FCF
Vocês são vencedores, vocês conseguem.

-----
Process exited after 4.371 seconds with return value 0
Pressione qualquer tecla para continuar. . . _
```

Figura 7 - Resultado da descriptografia do primeiro exemplo apresentado no trabalho.

[illegible]

Figura 8 – Resultado da descriptografia do segundo exemplo apresentado no trabalho.

```
C:\Users\User\Desktop\todos os testes de C\criptografia\Trabalhodescriptografia.exe
2
0
566F6388732073C66F2076656E6365646F867265732C00566F6388732073C66F2076656E6365646F867265732C00332C2C2C
Você são vencedores,
3
566F638873C320636F6E73656775656D2E002DC6C921B7B87FCF566F638873C320636F6E73656775656D2E002DC6C921B7B8
Você conseguem.

-----
Process exited after 6.294 seconds with return value 0
Pressione qualquer tecla para continuar. . . _
```

Figura 9 – Resultado da descriptografia do terceiro exemplo apresentado no trabalho.

Por fim, utilizando os exemplos disponibilizados dentro do trabalho, realizamos os testes da descryptografia dos hexadecimais fornecidos para testar se o código estava funcional e comparamos o resultado do nosso sistema com o resultado esperado e, após a análise, concluímos o projeto, pois os resultados estavam iguais aos esperados.

Discussão: Durante o projeto, surgiram alguns desafios. Um dos primeiros foi relacionado ao uso do scanf com o formato %2x. No início, planejávamos utilizar o código com apenas um array de unsigned char chamado “texto”, mas percebemos que alguns compiladores não funcionavam corretamente dessa forma. Para resolver esse problema e deixar o código compatível para vários compiladores, criamos um vetor do tipo int, chamado “textoNumero”, para armazenar os valores lidos em decimal e transferir eles para o array texto. Outra opção seria utilizar apenas o array do tipo int e acabar com o uso do array de char. No entanto, optamos por manter ambos os arrays, pois isso tornou o processo mais intuitivo facilitando o entendimento do código, pois ajuda a separar as etapas de leituras e processamento de dados.

Outro problema surgiu na etapa de escanear a mensagem hexadecimal. Durante a leitura, a variável `variavelB` estava sendo sobrescrita a cada chamada da função “`scanearMensagem`”. Para resolver isso, aumentamos o tamanho do vetor `texto` de 51 para 54 caracteres. Aparentemente, o problema ocorria porque o vetor `texto` ocupava um espaço de memória muito próximo ao da `variavelB` e, ao escrever no vetor, acabávamos sobrescrevendo a `variavelB`, redefinindo-a para 0, e por isso tivemos que aumentar o tamanho do vetor para evitar que ele interferisse na `variavelB`.

Conclusão:

Neste trabalho, praticamos diversos conceitos importantes na linguagem C, como manipulação de arrays, estruturas condicionais, laços de repetição, e lógica de programação em geral, além de aprimorar nossas habilidades em resolução de problemas. Desenvolvemos um código em C modularizado em cinco arquivos, sendo três com extensão “.c” e dois com extensão “.h”.

O objetivo final do código foi descriptografar mensagens de hexadecimal para decimal e remover caracteres indesejados. Como resultado, nosso sistema foi capaz de realizar essa tarefa com sucesso.

Este projeto foi essencial para fortalecer todo o nosso entendimento sobre a linguagem C e sobre resoluções de problemas, pelo fato de ter sido desafiador achar uma logica para desenvolver um projeto e ir moldando essa lógica e arrumando todos os problemas que foram aparecendo pela frente com diversas outras logicas diferentes.

Apêndice

<https://github.com/Kurok01/TrabalhoDeAlgoritmosEstruturaDeDados>