Aula 06:

Strings e Caracteres

Prof. Edvard

edvard@unifei.edu.br

Universidade Federal de Itajubá

Strings

- Uma **string** consiste em uma série de caracteres tratados como uma entidade única.
  - "**String**", em inglês, significa corda: um cordão que **une algumas coisas**.
  - Em C, uma string irá unir vários caracteres em uma única sequência, que possui algumas **características especiais** e pode se utilizar de **ferramentas próprias**.
  - Série ordenada de caracteres = array de caracteres. **Vetor**!
- Uma string pode incluir **letras**, **dígitos** numéricos e **caracteres especiais** como +, -, *, /, $, &.
  - Na verdade, uma string pode conter **praticamente qualquer coisa**. Sua flexibilidade na descrição da informação é que a torna importante.

Strings

- String é o nome que damos para uma sequência de caracteres armazenados de maneira sequencial na memória: **vetor de caracteres**.
- Sua declaração segue as mesmas regras de declaração de vetores convencionais:

```
char str[6]; // cadeia de caracteres
```

- Essa declaração cria uma string de nome **str** e tamanho **6**. No entanto, somente é possível armazenar uma palavra ou frase de até 5 caracteres.
  - Em C, as string precisam sempre terminar com o caractere nulo (‘\0’).

Strings

- O caractere nulo é necessário pois podemos definir uma string maior do que a palavra armazenada.
  - Imagine que você precisa de uma variável para guardar o nome de alguém, que pode ser pequeno como “José Silva” ou grande como “João Paulo Reus Rodrigues Leite”.
  - É necessário que a String seja declarada com um tamanho que caiba qualquer tipo de nome.

```
char nome[50]; // nome do usuário
```

- Ao preencher apenas as 10 primeiras posições do vetor com o nome “José Silva”, teremos outras 40 posições preenchidas com dados aleatórios da memória.
- Como o programa consegue identificar que a string termina no “Silva” e não prossegue com o lixo de memória?
  - Através do caractere nulo, que deve ser colocado após “José Silva”.

Strings

- Portanto, como o caractere ‘\0’ indica o final da string, devemos considerá-lo ao declarar seu tamanho.
  - Uma string de 50 posições somente suportará uma frase ou palavra de até 49 caracteres.
- A string “nome”, com o nome próprio “José Silva” ficaria da seguinte maneira:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	…	47	48	49
J	o	s	é		S	i	l	v	a	\0	ä	%	$	…	4	1	g

- E poderia ser inicializada de três modos:

```
char nome[50] = {'J', 'o', 's', 'e', ' ', 'S', 'i', 'l', 'v', 'a', '\0' };
char nome2[50] = "Jose Silva"; // Automaticamente adiciona \0 ao final
char nome3[] = "Jose Silva"; // Cria um vetor de exatamente 11 posições
```

Strings

- Todas as **propriedades** que aprendemos para **vetores** numéricos também podem ser aplicadas aos vetores de caracteres.
  - Por exemplo, podemos acessar individualmente cada caractere de uma string, tanto para saber seu valor quanto para modificá-lo:

```
char teste[] = "Teste";
printf("Valor inicial: %s \n", teste);

teste[0] = 'L';
printf("Valor final: %s\n", teste);
```

```
Valor inicial: Teste
Valor final: Leste

Process returned 0 (0x0)
Press any key to continue.
```

Strings

- Podemos utilizar as mesmas funções que já conhecemos para ler uma string do teclado ou escrever uma string na tela.
- **Escrita**: printf()
  - Utilizamos o código “%s” para identificar uma string.

```
char user[50] = "Theodomiro Santiago";
printf("O nome do usuario eh %s. \n", user);
```

```
O nome do usuario eh Theodomiro Santiago.

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```

- Repare que fazemos referência apenas ao identificador da string (**user**) e não utilizamos colchetes.

Strings e Caracteres

- O programa abaixo mostra a impressão de uma string completa e a impressão de alguns de seus caracteres separadamente. Repare na diferença:

```c
#include <stdio.h>

int main()
{
    char univ[] = "Universidade Federal de Itajuba";

    // Impressão do nome completo (%s)
    printf("Universidade: %s. \n\n", univ);

    // impressão de caracteres isolados (%c)
    printf("Primeiro Caractere: %c. \n", univ[0]);
    printf("Segundo Caractere: %c. \n", univ[1]);
    printf("Quinto Caractere: %c. \n", univ[4]);
    printf("Decimo Sexto Caractere: %c. \n", univ[15]);

    return 0;
}
```

```
Universidade: Universidade Federal de Itajuba.

Primeiro Caractere: U.
Segundo Caractere: n.
Quinto Caractere: e.
Decimo Sexto Caractere: d.
```

Strings

- **Leitura**: scanf()
  - Também utilizamos o código “%s” para identificar uma string.

```
// A variavel user pode conter uma string de até 19 caracteres + '\0'
char user[20];

// Repare que nesse caso não utilizamos o & antes do nome
// da variavel. O nome do vetor já representa o endereço
// de memória da primeira posição do vetor
scanf("%s", user);
```

- Também não utilizamos os colchetes no *scanf*. O nome da string é passado sozinho como parâmetro, sem o &.
- No entanto, o *scanf* pode não ser a melhor opção para a leitura de strings.

Strings

- A função scanf() lê apenas strings digitadas sem espaços, ou seja, lê apenas palavras.
  - Caso o usuário digite uma frase, ou um nome composto, apenas a porção anterior ao primeiro espaço será armazenada.
- Uma alternativa mais eficiente para leitura de uma string é a função **gets**(), a qual faz leitura dos dados de entrada considerando todos os caracteres digitados até que o usuário pressione a tecla "*Enter*".
  - Incluindo espaços.

```
char frase[50];

// Também não se utiliza & antes do nome da variável
gets(frase);
```

Strings

gets( )

stdio.h

```
char frase[50];

printf("Entre com uma frase: ");
gets(frase);
printf("Frase digitada: %s\n", frase);
```

```
Entre com uma frase: Universidade Federal de Itajuba
Frase digitada: Universidade Federal de Itajuba

Process returned 0 (0x0)   execution time : 47.710 s
Press any key to continue.
```

scanf()

stdio.h

```
char frase[50];

printf("Entre com uma frase: ");
scanf("%s", frase);
printf("Frase digitada: %s\n", frase);
```

```
Entre com uma frase: Universidade Federal de Itajuba
Frase digitada: Universidade

Process returned 0 (0x0)   execution time : 8.023 s
Press any key to continue.
```

Strings

A função **gets()**, entretanto, foi removida de c11, porque ela não controla o tamanho da string lida e pode resultar em estouro de memória (*buffer overflow*).
Sua utilização não é recomendada e pode resultar em avisos de problema no código (*warnings*).

```
  605 | extern char *gets (char *__s) __wur __attribute_deprecated__;
      |             ^~~~
/usr/bin/ld: /tmp/cc4trY63.o: in function `main':
main.c:(.text+0x37): warning: the `gets' function is dangerous and should not be used.
```

Strings

Uma alternativa é modificar o **scanf()** para ler a string incluindo os espaços:

```
scanf("%[^\n]s",str);
```

Os colchetes são chamados de scanset, que delimitam as informações lidas por **scanf().**
O símbolo **^** é um operador **OU EXCLUSIVO (XOR)**, que retorna true até que encontre um símbolo diferente.
A instrução **^\n** manda receber a entrada de dados, até que o caracter de nova linha **(\n)** seja encontrado.

Strings

```c
#include <stdio.h>

int main() {
    char str[100];
    int x;

    printf("Informe um inteiro: ");

    scanf("%d", &x);
    getchar();

    printf("Informe uma string: ");
    scanf("%[^\n]s",str);
    printf("%s",str);

    return 0;
}
```

scanf("%[^\n]s",str);

Essa construção funciona bem, porém repete os problemas de **scanf()** para caracteres, quando precedidos de outros **scanf()** para leituras de valores inteiros.

Quando for o caso, utilizar um **getchar()** antes de fazer a leitura de uma string resolve o problema.

Strings

A função mais recomendada é a **fgets()**, que tem a seguinte sintaxe:

```
fgets(NOME_STRING, TAMANHO, stdin);
```

Essa função toma controla o tamanho do *buffer*, portanto é mais segura do que **gets()**.

Strings

fgets(NOME_STRING, TAMANHO, stdin);

```c
#include <stdio.h>

int main() {
    char str[100];

    printf("Informe uma string: ");
    fgets(str, 100, stdin);

    printf("%s",str);

    return 0;
}
```

Strings

- Existem ainda algumas outras funções de entrada e saída de strings e caracteres definidas em stdio.h:
  - ***puts():*** *Imprime uma única string na tela, seguida de um caractere de fim de linha ('\n').*

```
printf("Frase Digitada: ");
puts(frase); // frase digitada + '\n'
```

  - ***getchar():*** *Permite ler um único caractere do teclado.*
  - ***putchar():*** *Permite imprimir um único caractere na tela.*

```
// le caractere ASCII, se for válido
letra = getchar();

printf("Caractere lido do teclado: ");
putchar(letra); // imprime único caractere
```

Strings

A função **fgets()** também tem um detalhe: ela inclui o "**\n**" ao final da string. Assim, na impressão pode haver diferenças.

Para remover o caractere de nova linha, pode-se usar a função **strcspn()**.

Essa função escaneia a primeira string até encontrar o primeiro caractere em comum com a segunda e retorna quantas posições foram percorridas. O retorno é um valor inteiro positivo. Sintaxe:

```
strcspn(str1, str2);
```

Usaremos logo após o **fgets()** da seguinte forma:

```
str1[strcspn(str1, "\n")] = 0;
```

Ao Trabalho!

- Escreva um programa simples, que receba uma string e a exiba na tela:

Ao Trabalho!

- Escreva um programa simples, que receba uma string e a exiba na tela:

```c
#include <stdio.h>

int main()
{
    char palavra[20];

    printf("Entre com uma string: ");
    fgets(palavra, 20, stdin);
    printf("String digitada: %s\n\n", palavra);

    return 0;
}
```

```
Entre com uma string: Itajuba
String digitada: Itajuba
```

Ao Trabalho!

- Escreva um programa que receba uma string e escreva na tela quantos caracteres a compõem, sem utilizar nenhuma função auxiliar pré-definida.

Ao Trabalho!

- Escreva um programa que receba uma string e escreva na tela quantos caracteres a compõem, sem utilizar nenhuma função auxiliar pré-definida.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char texto[51];
    int i;

    printf("Entre com um texto qualquer, com ate 50 caracteres: \n");
    fgets(texto, 51, stdin);

    texto[strcspn(texto, "\n")] = 0; //substitui o \n por zero no fim da string

    for(i = 0; i < 51; i++)
    {
        if(texto[i] == 0) //Encontrou o valor nulo, achou o fim da string
            break;
    }

    printf("\n\nO texto digitado possui %d caracteres\n\n", i);

    return 0;
}
```

```
Entre com um texto qualquer, com ate 50 caracteres:
Meu nome eh Jose da Silva e tenho 20 anos.


O texto digitado possui 42 caracteres
```

Ao Trabalho!

- Escreva um programa que receba uma string, e imprima o seu inverso na tela.

Ao Trabalho!

- Escreva um programa que receba uma string, e imprima o seu inverso na tela.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char palavra[20];
    int tamanho, i;

    printf("Entre com uma palavra qualquer: ");
    fgets(palavra, 20, stdin);

    palavra[strcspn(palavra, "\n")] = 0; /* remove \n da string */

    for(i = 0; i < 20; i++)
    {
        if(palavra[i] == 0) /* se for caractere nulo, terminou string*/
            break;
    }

    /* Tamanho da palavra */
    tamanho = i;

    printf("O inverso da palavra digitada eh: ");
    for(i = tamanho-1; i >= 0; i--)
    {
        printf("%c", palavra[i]);
    }
    printf("\n\n");

    return 0;
}
```

```
Entre com uma palavra qualquer: ENGENHARIA
O inverso da palavra digitada eh: AIRAHNEGNE
```

Strings

- Repare como seria interessante, nos dois problemas anteriores, que, por exemplo, houvesse alguma função pré-definida que nos **retornasse o tamanho da string** digitada.
- Existe uma série de funções pré-definidas na biblioteca padrão da linguagem C para o tratamento de strings. Elas estão definidas na biblioteca **<string.h>**.
- As mais utilizadas são:
  - ***strlen***: serve para obter o tamanho de uma string.
  - ***strcpy***: realiza uma cópia de uma string
  - ***strcmp***: compara strings
  - ***strcat***: concatena strings, ou seja, junta duas strings em uma só.

Strings

- **Tamanho de uma string | strlen**
  - Para se obter o tamanho de uma string (inteiro), utilizamos a função **strlen**(). A função irá retornar sempre a quantidade de caracteres que antecedem o caractere '\0'.
  - O programa anterior ficaria da seguinte maneira:

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char texto[51];
    int tamanho;

    printf("Entre com um texto qualquer, com ate 50 caracteres: ");
    fgets(texto, 51, stdin);

    texto[strcspn(texto, "\n")] = 0;

    tamanho = strlen(texto);

    printf("\n\nO texto digitado possui %d caracteres\n\n", tamanho);

    return 0;
}
```

```
Entre com um texto qualquer, com ate 50 caracteres:
UNIFEI


O texto digitado possui 6 caracteres
```

Strings

- **Copiando uma string | strcpy**
  - Uma string é um array de caracteres, e sabemos que a linguagem C não suporta a atribuição de um vetor para outro.
  - Por este motivo, a única maneira de atribuir o conteúdo de uma string para a outra é a **cópia, elemento por elemento**, de uma string para outra.
  - A linguagem C possui uma função que realiza esta cópia: **strcpy().**

    strcpy(char *destino, char *origem)

- **Ao Trabalho!**
  - Vamos escrever um programa que faça a cópia de duas strings elemento a elemento

Strings

- **Copiando uma string | strcpy**
  - Agora, vejamos como ficaria o código apenas utilizando a função strcpy.
  - O primeiro parâmetro é sempre a string de destino, e o segundo, a string de origem. Cerifique-se de que a string de **destino possui tamanho suficiente** para comportar a string de origem.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char texto[20], copia[20];
    int tamanho;

    printf("Entre com um texto qualquer, com ate 50 caracteres: ");
    fgets(texto, 20, stdin);

    texto[strcspn(texto, "\n")] = 0;

    strcpy(copia, texto);

    printf("\nO texto copiado: %s \n", copia);

    return 0;
}
```

```
Entre com seu nome: LECI e LECII
Nome digitado: LECI e LECII
Nome copiado: LECI e LECII
```

Strings

- **Concatenando uma string| strcat**
  - A concatenação consiste em copiar uma string para o final da outra, unindo as duas em uma única string.
  - É uma operação bastante frequente quando estamos lidando com processamento de informação textual.
  - O C possui uma função que realiza esta tarefa, a **strcat()**:

    strcat(char *destino, char *origem)

- **Ao Trabalho!**
  - Vamos escrever um programa que faça a concatenação de duas strings sem utilizar a função strcat.

Strings

- **Concatenando uma string| strcat**
  - Agora, vejamos como ficaria o código apenas utilizando a função strcat.
  - Neste caso, a string **destino precisa ter tamanho suficiente** para [illegible] strings destino e origem.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char destino[100], origem[50];

    printf("Entre o primeiro texto: ");
    fgets(destino, 100, stdin);
    destino[strcspn(destino, "\n")] = 0;

    printf("Entre o segundo texto: ");
    fgets(origem, 50, stdin);
    origem[strcspn(origem, "\n")] = 0;

    strcat(destino, origem);

    printf("\nO texto concatenado: %s \n", destino);

    return 0;
}
```

```
Entre com a primeira string: Engenharia
Entre com a segunda string: Eletronica
String concatenada: EngenhariaEletronica
```

Strings

- **Comparando strings| strcmp**
  - Da mesma maneira como o operador de atribuição não funciona corretamente para strings, o mesmo ocorre com operadores relacionais.
  - Se temos duas strings “str1” e “str2”, não é possível compará-las das seguintes maneiras:

    str1 == str2 | str1 != str2 | str1 > str2 | str1 < str2

  - É preciso utilizar a função **strcmp()**, que retorna**:**
    - **Zero**, se as strings forem iguais;
    - **Um valor maior que zero**, se str1 for alfabeticamente maior que str2;
    - **Um valor menor que zero**, se str1 for alfabeticamente menor que str2;

  - É importante lembrar que strcmp() é ***case sensitive***. Isso significa que letras maiúsculas ou minúsculas tornam as strings diferentes.

Strings

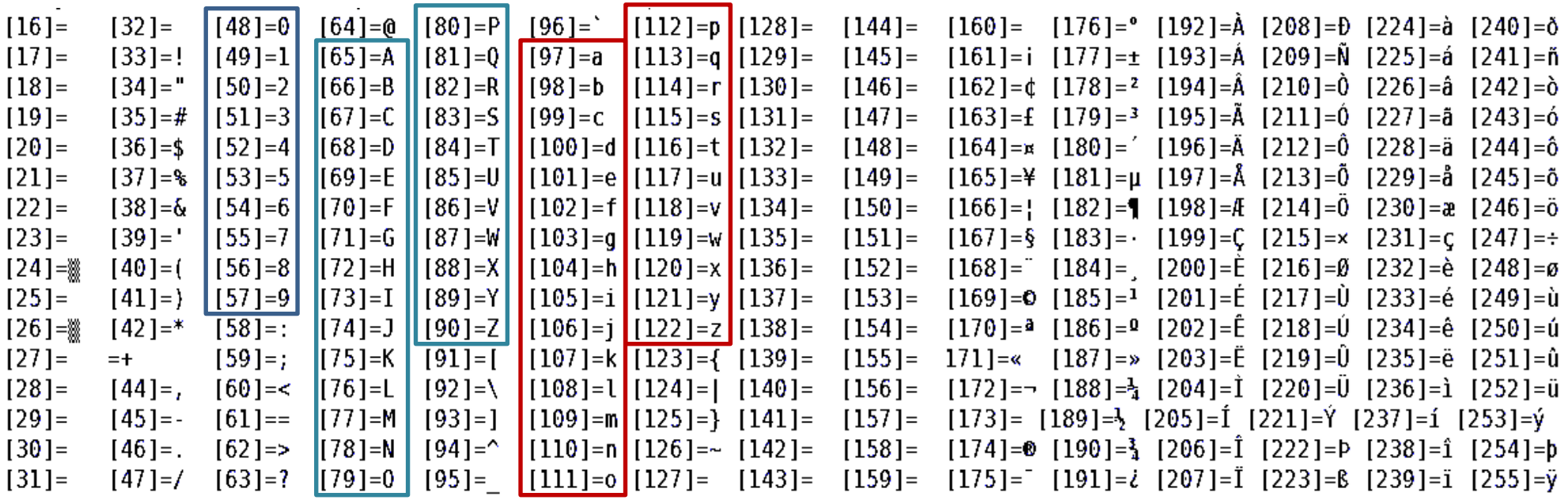
- **Comparando strings| strcmp**
  - Vejamos alguns exemplos:

str1	str2	strcmp(str1, str2)
“UNIFEI”	“UNIFEI”	0
“UNIFEI”	“UNIFAL”	1
“BRASIL”	“MEXICO”	-1
“Brasil”	“EUA”	-1
“joao”	“Joao”	1

A função strcmp() compara caractere a caractere, utilizando seus valores da tabela ASCII. Se não houver diferença, ela retorna 0. Se encontrar algum caractere diferente, ela retorna 1 se o valor for maior ou -1 se for menor.
QUASE uma ordem alfabética. (Quase? Maiúsculas e Minúsculas)

Strings

- **Comparando strings| strcmp**
  - Na tabela ASCII, os caracteres maiúsculos (65-90) vem antes dos minúsculos (97-122);
  - Os caracteres numéricos vem antes de qualquer letra.
  - São estes valores que são comparados na função strcmp().



Ao Trabalho!

Como ficaria, então, um programa que compara strings?

O que aconteceria se eu comparasse duas strings com o operador ==?

Ao Trabalho!

- Faça um programa que leia duas strings e as imprima em ordem alfabética, ou seja, a ordem em que apareceriam em um dicionário.

Ao Trabalho!

- Faça um programa que leia duas strings e as imprima em ordem alfabética, ou seja, a ordem em que apareceriam em um dicionário.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[20], str2[20];

    printf("Entre com a primeira string: ");
    fgets(str1, 20, stdin);
    printf("Entre com a segunda string: ");
    fgets(str2, 20, stdin);

    if(strcmp(str1, str2) == 0)
    {
        printf("Strings sao iguais.\n");
    } else {

        if(strcmp(str1, str2) > 0)
            printf("str1 > str2. \n");
        else
            printf("str1 <  str2. \n");

    }

    printf("Resultado: %d \n", strcmp(str1, str2));

    return 0;
}
```

Ao Trabalho!

- Escreva um programa em C que verifique se uma palavra é um palíndromo. **Exemplos**: ovo, arara, asa, osso, somamos, etc.
- Tente fazê-lo em casa!!

Ao Trabalho!

- Escreva um programa em C que verifique se uma dada palavra é um palíndromo. **Exemplos**.: ovo, arara, asa, osso, somamos, etc.

```c
#include <stdio.h>
#include <string.h>


int main()
{
    char palavra[50];
    int i, tamanho, is_it = 1;

    // Aquisicao dos dados
    printf("Entre com a palavra que gostaria de verificar: ");
    fgets(palavra, 50, stdin);

    palavra[strcspn(palavra, "\n")] = 0;

    tamanho = strlen(palavra);

    // processamento
    for(i = 0; i < tamanho/2; i++)
    {
        if(palavra[i] != palavra[tamanho-i-1])
        {
            is_it = 0;
            break;
        }
    }

    // imprime resposta
    if(is_it)
        printf("Palavra digitada eh um palindromo.\n\n");
    else
        printf("Palavra digitada NAO eh um palindromo.\n\n");

    return 0;
}
```

Ao Trabalho!

- O código de César é uma das técnicas de criptografia mais simples e conhecidas. É um tipo de codificação na qual cada letra do texto é substituída por outra, que se apresenta **n** posições após ela no alfabeto. Por exemplo, com uma troca de três posições, a letra A seria substituída por D, B se tornaria E, e assim por diante.
- Escreva um programa que faça uso deste código de César para **três posições**. Entre com uma string e imprima a string codificada. Exemplo:
  - String: “a ligeira raposa marrom saltou sobre o cachorro cansado”
  - Nova String: “d oljhlud udsrvd pduurd vdowrx vreuh r fdfkruur fdqvdgr”
- Tente fazê-lo em casa!!

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char str[100], strcod[100];
    int i;

    // recebe dados
    printf("Entre com a primeira string: ");
    fgets(str, 100, stdin);
    str[strcspn(str, "\n")] = 0;


    for(i = 0; i < strlen(str); i++)
    {
        if(str[i] == ' ')
            strcod[i] = ' ';
        else
        {
            strcod[i] = 97 + (str[i] - 97 + 3) % 26;
        }
    }

    strcod[i] = 0;

    printf("Nova String: %s. \n\n", strcod);

    return 0;
}
```