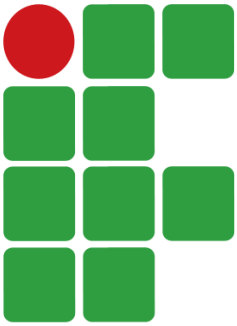

INSTITUTO FEDERAL
Mato Grosso
Campus Cuiabá - Octayde Jorge da Silva

LISTA 5 – FUNÇÕES LITERAIS

Aluno: Vitor Bruno de Oliveira Barth

Professor: Ruy de Oliveira

Disciplina: Algoritmos II

Cuiabá

2016

1 - Escreva um programa que leia várias linhas de texto, pelo teclado, e imprima uma tabela indicando o número de ocorrências de cada letra do alfabeto no texto.

```
algoritmo { lê uma frase e retorna as letras }
        declare frase[255] literal
        leia frase
        procuraLetras(frase)
fim-algoritmo

subrotina procuraLetras(ref frase literal) { subrotina irá verificar quais as letras contidas na frase }
        declare teste, letra[100], resultado[100][100] literal
        declare i, j, rletra[100] numérico
        i <- 0
        j <- 0
        para teste de “a” até “z”, faça { loop contará as letras contidas na frase }
                letra[i] <- teste
                rletra[i] <- 0
                para j de 0 até frase[j] = nulo faça
                        se teste = minusculo(frase[j]) então
                                nletras[i] <- nletras[i] + 1
                        fim-se
                fim-para
                imprima letras[i], rletras[i]
                i = i + 1
        fim-para
fim-subrotina
```

2 - Escreva um programa que leia várias linhas de texto, pelo teclado, e imprima uma tabela indicando o número de palavras com 1 letras, com 2 letras, 3 letras, etc., que aparecem no texto.

```
algoritmo { lê uma frase e retorna o tamanho das palavras }
        declare frase[255] literal
        declare i, j, tam, palavras[255][1] numérico
        leia frase
        para i de 0 até string[i] = nulo faça
                se string[i] = '_'  então
                        se palavras[i][0] = nulo então
                                palavras[i][0] <- 1 { i equivale ao tamanho e palavras[i][0] as repetições }
                        senão
                                palavras[i][0] <- palavras[i][0]+1
                        fim-se
                        i <- 0
                senão
                        i <- i + 1
                fim-se
        fim-para
        para i de 0 a palavras[i][0] = nulo faça
                imprima palavras[i][0], i
        fim-para
fim-algoritmo
```

3) Faça um programa que leia um vetor v de n elementos fornecido como entrada, e gere um vetor v2 contendo, na mesma ordem, somente os elementos não repetidos de v1. Ex: v1 = {2,4,2,6,7,9,6,5} v2 = {2,4,6,7,9,5}

```
algoritmo { esse algoritmo elimina os valores duplicados de um vetor }
        declare i, j numérico
        declare v1[255], v2[255], i literal
        declare teste booleano
        teste <- falso
        leia v1
        faça { verificará e o valor existe no vetor }
                para j de 0 até v2[j] = nulo faça
                        se v1[i] = v2[j]
                                teste <- verdadeiro
                        fim-se
                fim-para
        enquanto (v1[i] != nulo)
        se !teste { se não existir, será criada uma nova posição no v2 }
                v2[i] = v1[i]
        fim-se
        para i de 0 a v1[i] = nulo faça
                imprima v2[i]
        fim-para
fim-algoritmo
```

4 - Escreva um programa que leia várias linhas de texto, pelo teclado, e imprima uma tabela indicando o número de ocorrências de cada palavra distinta no texto. As palavras devem ser mostrada em ordem alfabética.

```
algoritmo { esse algoritmo indicará a quantidade de vezes que uma palavra se repete num texto }
        declare i, ocorrencias[255]  numérico
        i <- 0
        declare frase[255][255], palavras[255] literal { frase[][] será o texto e palavras[i] será a matriz das palavras,
sendo palavra[i] a palavra em si  e i o "codigo" da palavra em ocorrencias[]}
        faça
                leia frase[i][255];
                contePalavras(frase[i], palavras)
        enquanto(frase[i] != nulo) { a ultima linha deverá ser nula para parar a leitura de frases }
        imprimaAlfabetica(palavras, ocorrencias)
fim-algoritmo

subrotina contePalavras(frase[i numérico][] literal, ref palavras[] literal, ref ocorrencias numérico) { separará as
palavras na frase }
        declare j numérico
        declare palavraAtual literal
        palavraAtual <- ""
        para j de 0 a frase[i][j] = nulo faça
                se frase[i][j] != " " então { se o caracter não for espaço, palavraAtual será concatenará o prox char }
                        palavraAtual <- palavraAtual + frase[i][j]
                senão { se for espaço, colocará a palavraAtual na matriz de palavras e recomeçará PalavraAtual }
                        busquePalavra(palavraAtual, palavras, ocorrencias)
                        palavraAtual <- ""
                fim-se
        fim-para
fim-subrotina

subrotina busquePalavra(palavraAtual literal, ref palavras[] literal, ref ocorrencias[] numérico) { organizará as
palavras dentro da matriz de palavras }
        declare i numerico
        declare palavraEncontrada booleano
        palavraEncontrada <-false
        enquanto(palavras[i] = nulo) faça { busca na matriz palavras iguais }
                se palavraAtual <- palavras[i] então
                        ocorrencias[i] <- ocorrencias[i]+1
                        palavraEncontrada <- true
                fim-se
                i = i+1
        fim-enquanto
        se (!palavraEncontrada) então
                palavras[i] <- palavraAtual
                ocorrencias[i] <- 1
        fim-se
fim-subrotina
```

```
subrotina imprimaAlfabetica(palavras[] literal, ocorrencias[] literal)
        declare matrizImpressa[][] literal { criar uma matriz única juntando palavras[] e ocorrencias[] }
        para i de 0 até palavras[i] <- nulo faça
                matrizImpressa[i][0] <-palavras[i]
                matrizImpressa[i][1] <-ocorrencias[i]
        fim-para

        declare i, j numérico
        declare aux[1] literal
        para i de 0 até matrizImpressa[i][0] = nulo faça
                para j de 1 até matrizImpressa[i][0] = nulo faça
                        se (matrizImpressa[j][0] > matrizImpressa[j+1][0]) então
                                aux[0] <- matrizImpressa[j][0]
                                aux[1] <- matrizImpressa[j][1]
                                matrizImpressa[j][0] <- matrizImpressa[j+1][0]
                                matrizImpressa[j][1] <- matrizImpressa[j+1][1]
                                matrizImpressa[j+1][0] <- aux[0]
                                matrizImpressa[j+1][1] <- aux[1]
                        fim-se
                fim-para
        fim-para

        para i de 0 até matrizImpressa[i][0] = nulo faça
                imprima matrizImpressa [i][0], matrizImpressa[i][1]
        fim-para
fim-algoritmo
```

5- A datas são em geral impressas em vários formatos distintos em correspondências comerciais. Duas formatos muito comuns são 29/05/2014 29 de maio de 2014 Escreva um programa que leia a data no primeiro formato e a imprima no segundo formato

```
algoritmo { vai ler a data no formato dd/mm/aaaa e imprimir em forma de dd de mês de aaaa }
    declare dataLida[10], meses[12], mes literais
    leia dataLida
    meses = { Janeiro, Fevereiro, Março, Abril, Maio, Junho, Julho, Agosto, Setembro, Outubro, Novembro, Dezembro }
    se dataLida[3] = 1 então
        se dataLida[4] = 0 então
            mes <- 10
        senão se datalida[4] = 1 então
            mês <- 11
        senão
            mês <- 12
        fim-se
    senão
        mês <- dataLida[4]
    fim-se
    imprima dataLida [0], dataLida [1] de meses[mês] de dataLida [6], dataLida [7], dataLida [8], dataLida[9]
fim-algoritmo
```

6 - Escreva uma programa que leia uma frase e a imprima com a letra inicial de cada palavra em maiúsculo. Exemplo: "TrabaLhar é BOM" ◊ "Trabalhar É Bom."

```
algoritmo
        declare i, j numérico
        declare frase[255] literal
        leia frase
        para i de 0 ate frase[i] = nulo faça
                para j de 0 até frase[j] = ' ' { se tiver espaço, será reiniciada a contagem }
                        se j = 0 então
                                frase[i] <- maiusculo(frase[i])
                        fim-se
                j <- 0
        fim-para
        para i de 0 até frase[i] = nulo faça
                imprima frase[i]
        fim-para
fim-algoritmo
```

7 - Elabore um programa que leia um frase pelo teclado e a imprima com as letras em sequência de maiúsculo e minúsculo. Exemplo: “Lazer e SEMPRE necessário” ◊ “LaZeR é SeMpRe NeCeSsÁrIo”.

```
algoritmo
        declare frase[255] literal
        leia frase
        fminusculo(frase)
        formata(frase)
        para i de 0 até frase[i] = nulo faça
                imprima frase[i]
        fim-para
fim-algoritmo

subrotina fminusculo(ref frase literal)
        declare i numérico
        para i de 0 até frase(i) = nulo faça
                frase(i) <- minúsculo(frase[i])
        fim-para
fim-subrotina

subrotina formata(ref frase literal)
        declare i, j numérico
        i <- 0;
        j <- 0
        enquanto frase[i] != nulo faça
                j <- j + 1
                se frase(i) != ‘_’ && j < 2 então
                        frase[i] <- maiúsculo(frase[i])
                fim-se
                se frase[i] = ‘_’ ou j = 2 então
                        j <- 0
                fim-se
                i <- i + 1
        fim-enquanto
fim-subrotina
```

8 - Escreva um programa que leia uma frase e a escreva em sentido inverso e em maiúscula.

```
algoritmo
        declare frase[255] literal
        declare i, j numérico
        leia frase
        i <- 0
        enquanto frase[i] != nulo faça
                frase[i] <- maiusculo(frase[i])
        fim-enquanto
        para j de 0 até frase[j] = nulo faça
                imprima frase[i-j]
        fim-para
fim-algoritmo
```

9 - Elabore um programa que verifique se uma string é um palíndromo (equivalente ler da esquerda para a direita ou no sentido inverso).

```
algoritmo { esse algoritmo verificará se a a string digitada é palíndroma }
        declare frase[] literal
        declare tamanho numérico
        leia frase[]
        coloqueMinusculo(frase[], tamanho)
        se testePalindromo(frase[], tamano) então
                escreva "É palíndromo"
        senão
                escreva "Não é palíndromo"
        fim-se
fim-algoritmo

subrotina coloqueMinusculo(ref frase[] literal, ref tamanho numérico) { colocará a frase em minúsculo }
        declare i numérico
        para i de 0 até frase[i] = nulo faça
                minusculo(frase[1]) { sendo minusculo uma função pré definida da linguagem }
                tamanho <- i
        fim-para
fim-subrotina

subrotina testePalindromo(ref frase[] literal, tamanho numérico) { verificara se primeiro e último chars são iguais }
        declare i numérico
        declare teste booleano
        teste <- verdadeiro
        para i de 0 até tamanho/2 faça
                se frase[i] != frase[tamanho-i] && teste então { se o último for diferente do penúltimo e teste for F }
                        teste <- falso
                fim-se
        fim-para
        retorne teste
fim-subrotina
```

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

void coloqueMinusculo(char* frase, int *tamanho);
bool testePalindromo(char* frase, int tamanho);

void main() {
	char frase[255];
	int tamanho = 0;
	printf("Entre com a frase: ");
	scanf("%255[^\n]",frase);
	coloqueMinusculo(frase, &tamanho);
	if (testePalindromo(frase, tamanho))
		printf("\nPalindromo\n");
	else
		printf("\nNao Palindromo \n");
}

void coloqueMinusculo(char* frase, int *tamanho) {
	int i = 0;
	for (i = 0; frase[i]; i++) {
		frase[i] = tolower(frase[i]);
		*tamanho = i;
	}
}

bool testePalindromo(char* frase, int tamanho) {
	int i = 0;
	bool teste = true;
	for (i = 0; i < tamanho/2; i++) {
		if (frase[i] != frase[tamanho-i] && teste) {
			printf("%c diferente de %c", frase[i], frase[tamanho-i]);
			teste = false;
		}
	}
	return teste;
}
```

10 - Elabore um programa que receba o valor correspondente a 12 salários mensais, faça o somatório deles e apresente: o total de salários recebidos no ano, o maior salário e o menor salário. Utilize o código ASCII para uma melhor apresentação da mensagem para o usuário (com acentuação adequada).

```
algoritmo { esse algoritmo lerá o salário de 12 meses e apresentará informações sobre eles }
        declare salario[12], maiorSalario, menorSalario, totalSalario, i numérico
        leia salario[0] { para basearmos o maior e menor }
        maiorSalario <- salario[0]
        menorSalario <- salario[0]
        totalSalario = salario[0]
        para i de 1 a 10 faça
                leia salario[i]
                se salario[i] < menorSalario então { definirá o menor salário }
                        menorSalairo <- salario[i]
                fim-se
                se salario[i] > maiorSalario então { definirá o maior salario }
                        maiorSalario <- salario[i]
                fim-se
                totalSalario <- totalSalario + salario[i]
        fim-para
        imprima maiorSalario, menorSalario, totalSalario { usar tabela ASCII para imprimir caracteres especiais }
fim-algoritmo
```

```
#include <stdio.h>

void main() {
        float salario[12], maiorSalario, menorSalario, totalSalario;
        int i;
        printf("Insira o sal%crio do mes 1: ", 225u);
        scanf("%f", &salario[0]);
        maiorSalario = salario[0];
        menorSalario = salario[0];
        totalSalario = salario[0];
        for (i = 1; i < 11; i++) {
                printf("\nInsira o sal%crio do mes %i: ", 225u, (i+1));
                scanf("%f", &salario[i]);
                if (salario[i] < menorSalario)
                        menorSalario = salario[i];
                if (salario[i] > maiorSalario)
                        maiorSalario = salario[i];
                totalSalario += salario[i];
        }
        printf("\n\nO maior sal%crio foi %f\nO menor sal%crio foi %f\nO total recebido no ano foi %f\n", 225u,
maiorSalario, 225u, menorSalario, totalSalario);
}
```

11 - Escreva um programa que solicite ao usuário 10 números inteiros e, ao final, informe a quantidade de números ímpares e pares lidos. Calcule e mostre também a soma dos números ímpares e a média dos números pares. Use a tabela ASCII para melhorar a apresentação das mensagens para o usuário.

```
algoritmo { esse algoritmo lerá 10 números e dará informações sobre eles }
	declare numero, nImpares, nPares, mediaImpares, somaPares, i numérico
	nPares <- 0
	nImpares <- 0
	somaPares <- 0
	mediaImpares <- 0
	para i de 0 a 9 faça
		leia numero
		se numero%2 = 0 && numero != 0 então { ou seja, se ele for par }
			nPares <- nPares + 1
			somaPares <- somaPares + numero
		fim-se
		senão se numero != 0 então { se o número por impar}
			nImpares <- nImpares + 1
			mediaImpares <- mediaImpares + numero
		fim-se
	fim-para
	imprima nPares, nImpares, mediaImpares/nImpares, soma Pares { usar ASCII }
fim-algoritmo
```

```
#include <stdio.h>

void main() {
	int numero = 0, nImpares = 0, nPares = 0, somaPares = 0, i;
	double mediaImpares = 0;
	for (i = 0; i < 10; i++) {
		printf("\nInsira o n%cmero %i: ", 250u, i);
		scanf("%i", &numero);
		if (numero % 2 == 0 && numero != 0) {
			nPares++;
			somaPares += numero;
		}
		else if (numero != 0) {
			nImpares++;
			mediaImpares += numero;
		}
	}
	printf("\n\nForam inserido %i n%cmeros pares\nForam inseridos %i n%cmeros %cmpares\n", nPares, 250u,
nImpares, 250u, 237u);
	printf("\nA m%cdia dos n%cmeros %cmpares %c %f\nA soma dos n%cmeros pares %c %f\n", 233u, 250u,
237u, 233u, (mediaImpares/10), 250u, 233u, somaPares);
}
```