**Regras:**

1. Se este template não for submetido e preenchido corretamente, será atribuída a cotação zero ao laboratório;
2. Não se esqueça de submeter a pasta com o projeto do laboratório (código-fonte + makefile); sem código é atribuído zero ao laboratório.
3. Não “falsifique” os outputs dos níveis; alguma situação dessas resultará na atribuição de zero a dois laboratórios consecutivos.

# Identificação Alunos

* 190221093 | Alexandre Coelho
* 190221128 | Sérgio Veríssimo

# Nível 1 – Código de teste + output

**Output:**

Algorithm: strLength

input: [\*str - char array]

output: [size - int]

BEGIN

i <- 0

WHILE (str[i] ≠ '\0') DO

i <- i + 1

END WHILE

RETURN i + 1

END

ALGORITHM COMPLEXITY: O(n) -> LINEAR

Justificação:

Este algoritmo possui uma complexidade algorítmica O(n), porque o número de instruções cresce linearmente de acordo com o número de caracteres da string.

# Nível 2 – Código de teste + output

Output:

Algorithm: swapStrings

input: [\*str1 - char array, \*str2 - char array, by - integer]

output: [size - int]

BEGIN

str1Length <- strLength(str1)

str2Length <- strLength(str2)

tempChar <- ' '

IF (str1Length ≠ str2Length) THEN

RETURN false

END IF

IF (by > str1Length || by > str2Length || by <= 0) THEN

RETURN false

END IF

FOR i <- by - 1 TO (i < str1Length) DO

tempChar <- str1[i]

str1[i] <- str2[i]

str2[i] <- tempChar

i <- i + by

END FOR

RETURN true

END

ALGORITHM COMPLEXITY: O(n) -> LINEAR

Justificação:

Este algoritmo possui uma complexidade algorítmica O(n), porque é a soma de três lineares (O(n)) que dá O(3n). Como removemos a constante (3), fica O(n).

# Nível 3 – Código de teste + output

/\* T1 \*/

char str1 [] = "abcdefghij";

char str2 [] = "klmnopqrst";

bool test1 = swapStrings(str1, str2, 3);

printf("By: %d | S1: %s | S2: %s \n", 3, str1, str2);

/\* T2 \*/

char str3 [] = "abcdefghij";

char str4 [] = "klmnopqrst";

bool test2 = swapStrings(str3, str4, 1);

printf("By: %d | S1: %s | S2: %s \n", 1, str3, str4);

/\* T3 \*/

char str5 [] = "abcdefghij";

char str6 [] = "klmnopqrst";

bool test3 = swapStrings(str5, str6, 10);

printf("By: %d | S1: %s | S2: %s \n", 10, str5, str6);

/\* T4 \*/

char str7 [] = "abcdefghij";

char str8 [] = "klmnopqrst";

bool test4 = swapStrings(str7, str8, -5);

printf("By: %d | S1: %s | S2: %s \n", -5, str7, str8);

printf("\nT1: %s, T2: %s, T3: %s, T4: %s \n", test1 ? "true":"false", test2 ? "true":"false", test3 ? "true":"false", test4 ? "true":"false");

Output:

By: 3 | S1: abmdepghsj | S2: klcnofqrit

By: 1 | S1: klmnopqrst | S2: abcdefghij

By: 10 | S1: abcdefghit | S2: klmnopqrsj

By: -5 | S1: abcdefghij | S2: klmnopqrst

T1: true, T2: true, T3: true, T4: false

# Nível 4 – Código de teste + output

Vector vector1 = vectorCreate(4,6);

vectorPrint(vector1);

double vector1Length = vectorLength(vector1);

printf("\nVector1 Length: %f", vector1Length);

Vector vector2 = vectorCreate(-3,2);

vectorPrint(vector2);

double vector2Length = vectorLength(vector2);

printf("\nVector2 Length: %f", vector2Length);

int vectorDot = vectorDotProduct(vector1, vector2);

printf("\nVectorDot: %d", vectorDot);

int vectorDot = vectorDotProduct(vector1, vector2);

printf("\nVectorDot: %d", vectorDot);

Output:

Vector: X = 4 - Y = 6

Vector1 Length: 7.211103

Vector: X = -3 - Y = 2

Vector2 Length: 3.000000

VectorDot: 0

# Nível 5 – Código de teste + output

Vector vList1[] = { vectorCreate(4,6), vectorCreate(-3,2), vectorCreate(1,3) };

printf("\nvList1 exist orthogonals: %s", (existOrthogonals(vList1, 3) == 1) ? "true" : "false");

Vector vList2[] = { vectorCreate(4,6), vectorCreate(1,3), vectorCreate(0,2), vectorCreate(-1,5) };

printf("\nvList2 exist orthogonals: %s \n", (existOrthogonals(vList2, 4) == 1) ? "true" : "false");

Output:

vList1 exist orthogonals: true

vList2 exist orthogonals: false