EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 1 – De Python para C

Profa. Wu Shin-Ting

OBJETIVO: Introdução à programação em C com base em Python.

ASSUNTOS: Diferenças e semelhanças entre programação em Python e em C.

O que você deve ser capaz ao final deste experimento?

Distinguir uma linguagem de programação interpretada e uma linguagem compilada.

Perceber a semelhança da lógica do fluxo de execução de C e Python.

Distinguir tipagem dinâmica e tipagem estática.

Ter uma noção dos recursos de gerenciamento de memória em C.

Entender a relevância da programação em C para microcontroladores.

Gerar um código executável a partir de um código-fonte C no ambiente de desenvolvimento integrado (IDE) CodeWarrior.

Ter um primeiro contato com o ambiente IDE CodeWarrior.

INTRODUÇÃO

Na disciplina de Programação de micro- e mini-computadores foi introduzida a linguagem de programação Python. Nesta disciplina de Programação Básica de Sistemas Digitais precisamos nos inteirar com a linguagem C para programar os micro-controladores nos ambientes de desenvolvimento integrado IDE (*Integrated Development Environment*).

Linguagem interpretada e linguagem compilada

Python é uma **linguagem interpretada** de alto nível para propósito geral [1]. Ele foi desenvolvido por Guido van Rossum em 1989 com o objetivo de ter uma linguagem que apresenta uma sintaxe intuitiva, similar à linguagem natural em inglês, sem precisar se preocupar com a tipagem e o armazenamento de dados na memória como a linguagem C. A linguagem **C** é uma **linguagem compilada** de médio nível, também para propósito geral. Foi inventada por Dennis Ritchie em *Bell Laboratories* entre 1972--73 para programar sistemas operacionais que antes eram implementados com uma linguagem de baixo nível, o *assembly* [2]. O sistema operacional Unix dos minicomputadores como DEC DPD7 foi integralmente implementado com linguagem C. A relação entre C e Python é maior do que imaginamos. Há muita equivalência entre as sintaxes das duas linguagens, a menos dos operadores relacionados com o uso da memória.

Uma **linguagem compilada** é uma linguagem que requer que uma máquina converta, por uma cadeia de ferramentas (*toolchain*), os códigos de um programa, denominados **códigos-fonte**, em códigos binários da máquina, conhecidos por **códigos executáveis**, antes da sua execução. Uma **linguagem interpretada** é uma linguagem para a qual a máquina traduz, em tempo de execução, as suas instruções para as referências às funções pré-implementadas (*built-in functions*) e os valores dos seus argumentos. Por dispensar a interpretação dos códigos em funções pré-compiladas, o tempo de execução de um programa compilado é menor do que o tempo de execução de um

programa interpretado. Um programa compilado apresenta um melhor desempenho temporal. Portanto, a linguagem compilada é ainda a preferida em aplicações relacionadas com o *hardware* quando o tempo é um fator crítico, como sistemas operacionais, *drivers* e *firmwares*. Por outro lado, partindo da premissa de que todas as funções pré-implementadas estejam devidamente testadas, os erros dos programas interpretados se limitam aos erros detectados durante a interpretação das suas instruções no momento da execução. Assim, a linguagem interpretada tem sido a preferida para prototipagem e provas de conceito no desenvolvimento de um projeto.

Na Seção 2 (página 3) em [3] é apresentada de forma comparativa as estruturas básicas de um programa em Python e em C. As linhas de instrução em Python só contêm os comandos de execução para serem interpretados no tempo de execução, enquanto as linhas de instrução em C podem contem as diretivas do pré-processador (Figura 1) que são transformadas em comandos nativos de C antes de serem compilados junto com os comandos de instrução em códigos de máquina. Nas Tabelas 1 e 2 da Seção 3 em [3] são mostradas comparativamente os operadores e os comandos de Python e C. Uma breve explicação sobre diretivas é dada na Seção 3.1 (página 5) em [3].

Tipagem de dados

Em ambas as linguagens, as unidades de armazenamento de um espaço de memória, em *bytes*, são definidas por endereços (por *bytes*) e conteúdos/valores guardados a partir desses endereços, podendo ocupar um ou mais *bytes*. Essas unidades são denominadas **variáveis** (Seção 4/página 11 em [3]). A linguagem C dispõe de uma série de recursos para um programador gerenciar o uso da memória de forma dedicada. Através das **declarações de variáveis**, programadores atribuem um nome a uma unidade de memória alocada e define o tipo de dado a ser armazenado nela. Dizemos que a **tipagem** de dados em C é **estática**, porque t**odos os dados** utilizados num programa precisam ter seus espaços de memória alocados e seus tipos definidos antes do uso. Os tipos de dados básicos em C são: **char** (caractere), **int** (valor inteiro), **float** (valor em ponto flutuante), **double** (valor em ponto flutuante duplo) e **void** (sem valor). Em particular, o tipo **void *** é reservado para declarar o endereço de memória de um tipo qualquer. Para a declaração de um endereço de memória de um tipo de dado específico *tipo*, usa-se ***tipo ****. São ainda disponíveis em C um conjunto de **qualificadores/modificadores** de acesso, do tamanho e do sinal desses tipos básicos (Seção 4.1/página 11 em [3]). Nos projetos de sistemas embarcados é muito comum usar os tipos de dados definidos no arquivo `stdint.h` pelo padrão ISO C99 [19], por especificarem explicitamente o tamanho e o sinal dos valores armazenados (Seção 4.1/página 12 em [3]). Por exemplo, para a seguinte declaração

```
uint16_t a;
```

o compilador aloca para a variável `a` um espaço de endereço de 16 *bits* (2 *bytes*) e gera instruções que interpreta o código binário armazenado neste espaço como um valor inteiro sem sinal.

A linguagem Python, por sua vez, abstrai essas unidades de armazenamento em **objetos** e dispõe de funções pré-implementadas que alocam e desalocam espaços de memória de forma transparente para programadores. Além dos valores propriamente ditos de uma variável, um objeto contém as funções que possam ser aplicadas sobre esses valores. Os tipos de dados das variáveis são automaticamente definidos no momento de atribuição de valores a elas. Daí a denominação de

tipagem dinâmica no Python. Usar espaços de memória sem se preocupar com as declarações das variáveis é uma vantagem oferecida pelo Python. Por outro lado, o Python não oferece recursos adicionais para implementar soluções alternativas, como alocar 1, ao invés de 4 *bytes*, para representar valores inteiros entre 0 a 255. Isso pode ser um problema para dispositivos com recursos escassos como microcontroladores.

Escopo das Variáveis

Em termos de acessibilidade, tanto C quanto Python diferenciam as variáveis em locais e globais. As **variáveis locais** são apenas visíveis/acessíveis dentro do escopo de instruções em que elas são definidas, enquanto as **globais** são acessíveis por um escopo maior podendo abranger um arquivo ou um módulo de funções (Seção 4.2/página 15 em [3]). Diferente de Python, as variáveis locais e globais em C tem diferentes tempos de existência na memória. As variáveis locais são extintas da memória assim que se conclui a execução do escopo das instruções em que elas são válidas, enquanto as variáveis globais tem o seu endereço preservado até o final de execução de um programa.

Gerenciamento de memória

Uma grande diferença entre C e Python está no **gerenciamento de memória**. O Python assume a função de **coleta de lixo** (*garbage collector*) de unidades de memória. Isso simplifica a implementação de qualquer algoritmo, principalmente aquele que envolve relações mais complexas de dados. Por outro lado, visando a atender aplicações de múltiplos propósitos, as soluções consideradas otimizadas pelos seus desenvolvedores nem sempre são as melhores para uma tarefa específica. Em C, o gerenciamento de memória é manual. Através das funções de alocação de memória dinâmica, *malloc*, *realloc*, *calloc* e *free* [9], o programador aloca em tempo de execução do programa unidades de memória necessárias e libera as que não são mais requisitadas para reuso. No entanto, em aplicativos embarcados, onde a previsibilidade das respostas e o tempo de execução das suas instruções são fatores críticos, deve-se evitar a prática de alocação dinâmica.

Passagem de parâmetros para as funções

Uso de funções evita a repetição de instruções/códigos dentro de um mesmo programa e ajuda na modularização e no reuso dos códigos. Muitas funções requerem dados de entrada e retornam dados de saída. Esses dados são passados como argumentos/parâmetros. No Python a passagem é **por referência** (o endereço). Por compartilharem uma função e a função que a chamou o endereço de uma variável, as modificações nos valores da variávei feitas na função é refletida na função que a chamou. Em C, a passagem é **por valor** (o conteúdo), ou seja, a função recebe uma cópia do valor passado. Portanto, as modificações no valor dentro da função não são refletida na função que a chamou. Para poder atualizar dentro de uma função o valor de uma variável da função que a chamou, é necessário passar como "valor" o endereço da variável. Em C, distingue-se o valor de uma variável e o seu endereço pelo operador "endereço-de" **&**. Quando o valor da variável é um endereço, distingue-se o seu valor (endereço) e o seu conteúdo pelo operador "valor-de" ***** (Seção 4.3/página 17 em [3]).

Comunicação com o mundo externo

A forma como uma máquina se comunica com o mundo externo, assistindo as tarefas humanas, é através das **funções de entrada e de saída**. Através da entrada, um programa recebe os dados

necessários para a sua execução, transferindo-os dos periféricos de entrada para memória. E através da saída, o programa envia para os periféricos os resultados computados, fazendo a transferência dos dados da memória para os periféricos. Tipicamente, um usuário fornece os dados via teclado e o programa envia os resultados para a tela. As funções de entrada e saída básicas no Python, *input* e *print*, são disponíveis por padrão [10]. Enquanto as funções de entrada e saída básicas de C, *scanf* e *printf*, estão numa biblioteca-padrão cujo arquivo-cabeçalho *stdio.h* precisa ser incluído para usá-las [11]. Essas funções operam sobre *streams* (sequências) de caracteres e são gerenciadas pelo **sistema operacional**. Além das variáveis de entrada/saída, são inclusos entre os argumentos dessas funções os formatos como tais *streams* de caracteres devem ser interpretados.

Conversão de códigos-fonte em códigos executáveis

Figura 1 mostra as ferramentas envolvidas na conversão de códigos em linguagem C armazenados em arquivos de extensão *.c* num arquivo executável de extensão *.elf*: **pré-processador** para traduzir as **diretivas** de C em arquivos de extensão *.i* com instruções puras de C; **compilador** para traduzir as instruções puras em C em arquivos-objeto de extensão *.o* contendo códigos de máquina do processador-alvo, e **ligador** para juntar as instruções de diferentes arquivos e construir um arquivo executável de extensão *.elf*. Além dos erros durante a execução do programa, podem ocorrer erros em cada estágio de um *toolchain*. O **diagnóstico dos erros** em cada estágio nem sempre é uma tarefa simples.

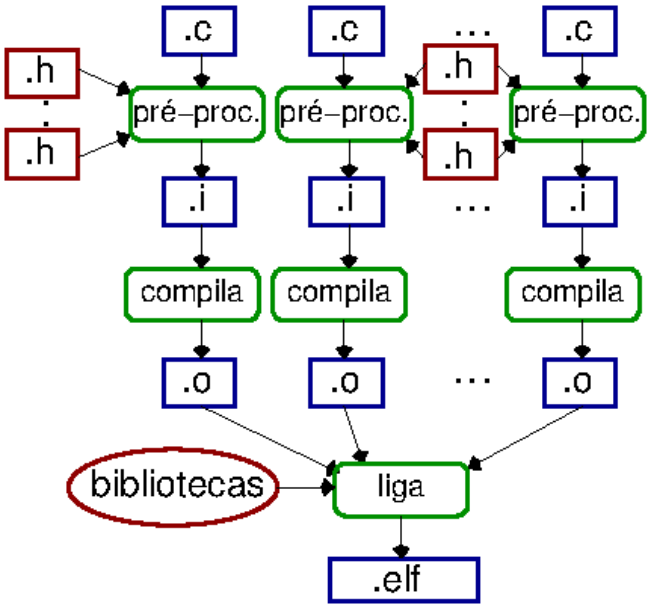

Figura 1: Processo de geração de um executável a partir de um código-fonte em C.

Nesta disciplina usaremos o **ambiente de desenvolvimento integrado (IDE) CodeWarrior**, que tem uma interface gráfica para assistir ao desenvolvimento de aplicativos embarcados nos microcontroladores NXP Kinetis.

Hardware, Firmware e Software

O conjunto de componentes físicos, circuitos integrados, dispositivos eletrônicos, placas, monitor, equipamentos periféricos etc., que forma um sistema digital (computador) é denominado ***hardware*** do sistema. O conjunto de instruções que controlam diretamente a operação de *hardware*, complementando as suas funções, é denominado ***firmware***. Exemplos de *firmware* são BIOS (*Basic*

Input/Output System) e UEFI (*Unified Extensible Firmware Interface*). Os *drivers* são *firmwares* que controlam dispositivos específicos de *hardware*, como as placas de rede, vídeo e som, conectando-os com o **sistema operacional**. ***Software*** é um conjunto de instruções para um processador de um sistema digital controlar todos os *hardwares* reconhecidos por ele. O sistema operacional é um *software*. A maioria dos *softwares* com os quais estamos familiarizados, como navegador e editor de textos, é chamada de ***software* de aplicativo**. Ao contrário do *hardware*, o *software* é extremamente flexível, uma vez que permite ser continuamente atualizado e alterado sem que o sistema seja resetado.

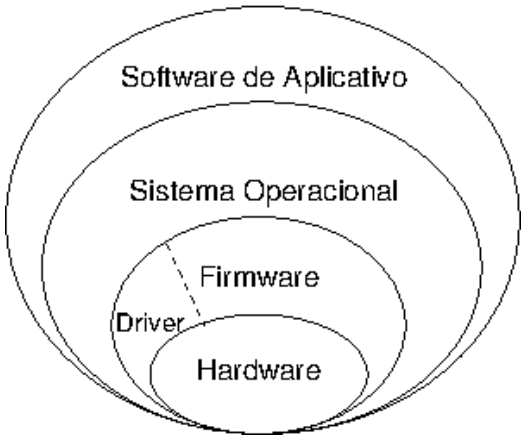

Figura 2: *Hardware*, *firmware*, *driver* e *software.*

O alvo desta disciplina são os microcontroladores desprovidos de um sistema operacional. Estaremos programando diretamente sobre o *hardware*, ou seja, sobre o *bare-metal* (metal nu). Os programas são denominados ***bare-metal firmware*** (*firmware* em metal nu). Veremos como configurar um *hardware* através de um programa em C e programar, também em C, o processador para executar uma tarefa específica sobre o *hardware* configurado.

Informações Adicionais

Convido fazer uma visita ao tutorial interativo *online* de C [4] e tentar fazer os exercícios propostos para ganhar familiaridade com C.

EXPERIMENTO

Neste experimento vamos nos familiarizar com a linguagem de programação C através dos conhecimentos prévios de Python. São usados o interpretador Python [5], o compilador C *online* [6] e o ambiente IDE CodeWarrior instalado nos computadores do LE-30.

1. Analise os programas em Problema 12, em Python e C, do projeto MAC Multimídia – inteiros [7]; em Problema 3 do projeto MAC Multimídia – reais [8], em C [13] e em Python [14]; e em Problema 6 do projeto MAC Multimídia – vetores [22], em C [15] e em Python [16].

1.a **Tipagem de Dados e Escopo das Variáveis:** Identifique as variáveis em Python e os seus equivalentes em C na tabela abaixo. Identifique ainda o tipo de dado e a visibilidade/contexto para cada uma delas. Há alguma variável global nos 3 programas?

1.b **Funções de Entrada e Saída:** Considerando como variáveis de trabalho todas as variáveis que não sejam de entrada nem de saída, classifique na tabela abaixo as variáveis identificadas

em entrada/saída/trabalho. Atenta aos operadores & na passagem de variáveis em funções de entrada nos códigos em C.

1.c **Quantidade de *bytes* alocada para variáveis:** Nas soluções do Problema 3 é incluída uma instrução de verificação do espaço de memória alocado para as variáveis (`sizeof`). Em qual das duas linguagens o tamanho alocado é maior para uma mesma variável? Justifique.

1.d **Leitura de um código em C:** Em [7] são apresentadas 3 soluções para o Problema 12 em Python e 1 solução em C. Analise-as. Identifique e justifique qual deles é implementado na solução em C.

	Python (tipo de dado, visibilidade)	C (tipo de dado, visibilidade)	Entrada/Saída/de Trabalho
Problema 12			
Problema 3			
Problema 6			

2. **Operadores *bit*-a-*bit*:** Para otimizar o uso da memória, operações *bit*-a-*bit* são aplicadas na compactação e na extração de informações nos registradores de um microcontrolador. O par de programas `bit_op.*` [17], [18] demonstra o uso de operadores *bit*-a-*bit* definidos em C e Python. Em C as variáveis a, b e c são declaradas como do tipo `uint8_t`, para o qual o compilador aloca um espaço de memória de 8 *bits* e interpreta todos os valores armazenados nesses 8 *bits* como um inteiro sem sinal. Analise de forma comparada os dois programas com diferentes valores de entrada para os operandos `a` e `b`.

2.a Preencha em cada linha da tabela abaixo o nome de cada operador e os resultados obtidos para a = 100 e b = 100.

2.b Explique a diferença nos resultados das operações ~a e ~b no programa C. Dica: Ver a solução do item 4.

	Nome do Operador	Python	C
a & b			
a | b			
a ^ b			
~a			
~b			
a << 2			
a >> 2			

3. **Criação e execução de projetos sem funções de entrada/saída:** As funções de entrada (*scanf*) e saída (*printf*) são implementadas em cima de um sistema operacional. As bibliotecas-padrão C instaladas no ambiente IDE CodeWarrior não dispõem essas duas funções. Portanto, a partir dos programas em C dados nos itens 1 e 2, o ligador não encontrará a definição das duas funções e não conseguirá gerar os respectivos executáveis. Podemos, porém, inserir os valores diretamente nos endereços das variáveis pela aba `Variables` da perspectiva `Debug` do IDE ao forçarmos a parada do fluxo de controle do programa por meio de *breakpoints*. Pois, quando se pára o fluxo de controle numa linha de comando, todas as instruções antes da linha foram executadas e o processador aguarda a execução da linha de parada. Se inserirmos valores em variáveis/endereços antes de continuar a execução, os valores inseridos serão considerados. Por exemplo, com a seguinte adaptação do programa `fatorial.c` [20], podemos gerar um projeto executável no nosso microcontrolador e variar o valor da variável de entrada `n`, assim como ler o valor do resultado `res`, se colocarmos na função `main` um *breakpoint* na linha da chamada `fatorial (n, &res)` (antes da execução da função para inserir um valor em `n`) e um na linha seguinte (após a execução para ver `n!` em `res`):

```
/*
 * fatorial.c
 */

/* Bloco de diretivas de pre-processamento */
//#include <stdio.h>

//#include "stdint.h"

/* Bloco de prototipos das funcoes */
int fatorial(int n, int *fatorial);

/* Definicao da funcao main */
int main (){
  int n;        /* guarda o numero dado */
  int    res;

  for (;;) {
//  printf("\n\tCalculo do fatorial de um numero\n");
//  printf("\nDigite um inteiro nao-negativo: ");
//
//  scanf("%d", &n);

  fatorial(n, &res);

//  printf("O valor de %d!: %d\n", n, res);
  }

  return 0;
}

/* Definicao da funcao fatorial */
int fatorial(int n, int *fatorial) {
  int contador;

  /* inicializacoes */
  *fatorial = 1;
  contador = 2;

  while (contador <= n) {
    *fatorial = *fatorial * contador;
    contador = contador + 1;
  }

  return *fatorial;
}
```

Siga as instruções na referência [21] para criar um novo projeto no ambiente IDE CodeWarrior (Seção 2.1, página 4), para gerar um executável (Seção 2.3, página 18), para transferir o executável gerado no PC para o microcontrolador (Seção 2.4, página 24), para criar *breakpoints* na perspectiva `Debug` (Seção 2.5.1, página 28) e para acessar os endereços e os valores das variáveis declaradas na Aba `Variables` (Seção 2.5.3, página 31). Antes da geração do executável, substitua o conteúdo do arquivo `main.c` pelo `fatorial.c` (Seção 2.2.1, página 12). Após a geração do executável de `fatorial`, identifique entre as linhas de mensagem mostradas na aba `Console` os arquivos de extensão .o e `.elf` (Figura 1). A ordem em que esses arquivos foram gerados é condizente com a sequência mostrada na Figura 1?

4. **Impacto do tipo de dados nos resultados:** O processador do nosso microcontrolador é de 32 *bits*. Todas as operações lógico-aritméticas são em 32 *bits*. Quando os operandos são variáveis de tipos de dados de tamanho menor, eles são automaticamente estendidos para 32 *bits*. Vamos analisar o impacto da capacidade de armazenamento de um espaço alocado a uma variável e do seu tipo de dado nos resultados de uma operação aritmética usando aba `Variables` da perspectiva `Debug`. Modifique, **de forma consistente**, o tipo de dado da variável `res` da função `main` de `fatorial` para os tipos **uint8_t** (inteiro sem sinal de 8 *bits*), **uint16_t** (inteiro sem sinal de 16 *bits*), **uint32_t** (inteiro sem sinal de 32 *bits*), **int8_t** (inteiro com sinal de 8 *bits*), **int16_t** (inteiro com sinal de 16 *bits*) e **int32_t** (inteiro com sinal de 32 *bits*). Uma modificação consistente implica em que as variáveis `res`, `fatorial` e a função `fatorial` destacadas no código do item 3 sejam declaradas com o mesmo tipo de dados**. Sendo C uma linguagem compilada, deve-se refazer "*Build Project*" para cada modificação no código-fonte** (Seção 2.3/página 18 em [3])**.** Anote os resultados armazenados no endereço `res`, na base hexadecimal/binária, para n=5, 6, 7, 8, 9, 10, 11, 12, 13.

   4.a Para cada tipo de dado (coluna), qual é o maior valor `n` para o qual os resultados gerados são corretos?

   4.b Por quê os tipos de dados sem sinal (`uint8_t`, `uint16_t` e `uint32_t`) conseguem cobrir mais resultados corretos de `n!` do que os tipos de dados com sinal (`int8_t`, `int16_t` e `int32_t`)?

   4.c Explique sucintamente como os tipos de dados impactam nos resultados gerados embora o *hardware* seja o mesmo para todas as operações executadas.

	uint8_t	uint16_t	uint32_t	int8_t	int16_t	int32_t
n=5						
n=6						
n=7						
n=8						
n=9						
n=10						
n=11						
n=12						
n=13						

RELATÓRIO

O relatório deve ser devidamente identificado, contendo a identificação do instituto e da disciplina, o experimento realizado, o nome e RA do aluno. Para este experimento, elabore um documento, no **formato pdf**, contendo as respostas dos itens 1 a 4 do roteiro. Suba-o no sistema *Moodle*.

REFERÊNCIAS

[1] Guido van Rossum. An Introduction to Python for Unix/C Programmers.
Proc. of the NLUUG najaarsconferentie. Dutch UNIX users group}, 1993.
[2] Brian W. Kernighan and Dennis Ritchie. C Programming Language. *Prentice Hall*, Primeira edição, 1978.
[3] Wu Shin Ting. Referência Comparativa Rápida entre Python e C para Sistemas com Recursos Limitados.
https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos_ea871/Python_C.pdf
[4] learn-c.org. Interactive C tutorial. https://www.learn-c.org/
[5] Interpretador de Python *online*. https://www.online-python.com/
[6] Compilador de C *online*. https://www.onlinegdb.com/online_c_compiler
[7] USP – IME. Projeto MAC Multimídia: inteiros.
https://www.ime.usp.br/~macmulti/exercicios/inteiros/index.html
[8] USP – IME. Projeto MAC Multimídia: reais
https://www.ime.usp.br/~macmulti/exercicios/reais/index.html
[9] Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc()
https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/
[10] Python: Input/Output Functions
https://www.decodejava.com/python-input-output.htm
[11] C Input and Output - printf()/scanf(), and more
https://www.studytonight.com/c/c-input-output-function.php
[12] Aprendizado de Algoritmos usando o Portugol Studio
https://www.academia.edu/33541675/Aprendizado_de_Algoritmos_usando_o_Portugol_Studio
[13] pertence.c
https://www.dca.fee.unicamp.br/cursos/EA871/2s2022/codes/pertence.c
[14] pertence.py

https://www.dca.fee.unicamp.br/cursos/EA871/2s2022/codes/pertence.py
[15] ocor_palavras.c
https://www.dca.fee.unicamp.br/cursos/EA871/1s2022/codes/ocor_palavras.c
[16] ocor_palavras.py
https://www.dca.fee.unicamp.br/cursos/EA871/1s2022/codes/ocor_palavras.py
[17] bit_op.py
https://www.dca.fee.unicamp.br/cursos/EA871/2s2022/codes/bit_op.py
[18] bit_op.c
https://www.dca.fee.unicamp.br/cursos/EA871/2s2022/codes/bit_op.c
[19] stdint.h
https://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdint.h.html
[20] fatorial.c
https://www.dca.fee.unicamp.br/cursos/EA871/1s2022/codes/fatorial.c
[21] Ambiente de Desenvolvimento – *Software*
https://www.dca.fee.unicamp.br/cursos/EA871/references/apostila_C/AmbienteDesenvolvimentoSoftware_V1.pdf
[22] USP – IME. Projeto MAC Multimídia: vetores.
https://www.ime.usp.br/~macmulti/exercicios/vetores/index.html

Revisado em 28/12/2022
Revisado em 07/08/2022
Elaborado em 23/2/2022