EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 1 – De Python para C

Profs. Antonio Augusto Fasolo Quevedo e 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 lingagem compilada.

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

Distinguir tipagem dinâmica e tipagem estática.

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

Entender a relevância da programação em C para microcontroladores.

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*).

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 em códigos binários da máquina 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.

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. Assim, a linguagem interpretada tem sido a preferida para prototipagem e provas de conceito no desenvolvimento de um projeto.

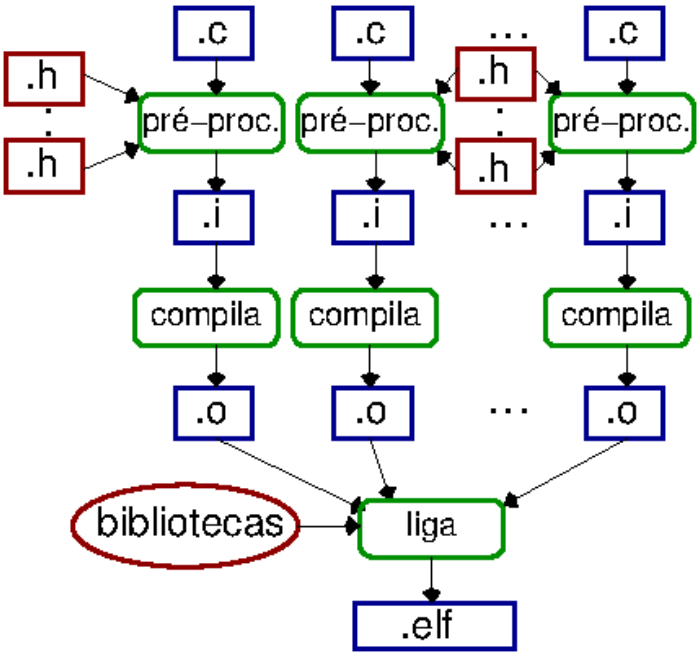

Figura 1: Processo de geração de um executável a partir de um código-fonte em C.

Uma grande diferença entre C e Python está no **gerenciamento de memória**. Em ambas as linguagens, as unidades de armazenamento de um espaço de memória, em *bytes*, são abstraídas em endereços e conteúdos/valores dessas unidades. A linguagem C dispõe de uma série de recursos para o programador gerenciar o uso da memória de forma dedicada, enquanto Python abstrai essas unidades de armazenamento em objetos e dispõe de funções pré-implementadas que alocam e dealocam a memória, de forma transparente para programadores. O fato do Python assumir a função de coleta de lixo (*garbage collector*) de unidades de memória 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. 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.

Uma comparação sucinta entre Python e C pode ser encontrada em [3]. Recomendo a sua leitura antes de iniciar a resolução das atividades deste experimento. Convido ainda para 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] e o compilador C online [6].

1. **Declaração de variáveis:** Todas as variáveis tem um endereço na memória onde os seus valores são armazenados. Em C elas devem ser explicitamente declaradas para que o compilador aloque um espaço de memória apropriado (**tipagem estática**), enquanto em Python essa escolha é feita de acordo com o valor atribuído à variável em tempo de execução (**tipagem dinâmica**). Identifique as variáveis em Exercício13, em Python e C, do projeto MAC Multimídia [7], em programas de pertinência de um ponto a um disco unitário, pertence.py e pertence.c [8], e de contagem de ocorrências de uma palavra numa frase, ocor_palavras.py e ocor_palavras.c. Para os programas em C, identifique também o tipo de valor (caracter, inteiro ou ponto flutuante) e o tamanho do espaço de memória declarados para cada variável. Organize, em linhas, os pares de variávieis por programa. **Dica:** Use a função `sizeof` para determinar o espaço de memória alocado para cada variável em C, como ilustra o código pertence.c.

	Python	C (tipo e tamanho do conteúdo em *byte*)	Entrada/Saída/de Trabalho
Exercício 13			
pertence.*			
ocor_palavras.*			

2. **Passagem de valores entre funções:** Através das funções de entrada (dos valores das variáveis) e de saída (dos resultados computados), pode-se variar a tarefa de um programa. Os dados são passados como argumentos dessas funções. Em C, a passagem é **por valor** (o conteúdo) e em Python, **por referência** (a referência ao endereço). Porém, para atualizar o valor de uma variável com a entrada, é necessário passar como "valor" o endereço da variável. Em C, distingue-se o (valor de) conteúdo e o (valor de) endereço pelo operador "endereço de" **&**. Observe nos programas em C do item (1) o uso desse operador em `scanf`. Classifique na coluna "Entrada/Saída/de Trabalho" da tabela do item (1) o papel de cada variável declarada nos 3 programas.

3. **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 programaas `bit_op.*` demonstra o uso de operadores *bit*-a-*bit* definidos em C e Python. Analise de forma comparada os dois programas com diferentes valores de entrada para os operandos `a` e `b`. Preencha em cada linha da tabela abaixo o nome de cada operador, justificando sucintamente os resultados obtidos para a = 87 e b = 60.

	Nome do Operador	Python	C

a & b			
a | b			
a ^ b			
~a			
~b			
a << 2			
a >> 2			

4. **Nível de abstração:** C possui uma sintaxe que se aproxima do repertório de instruções da máquina (linguagem de médio/baixo nivel), enquanto Python apresenta uma sintaxe mais intuitiva com tais instruções sintetizadas em comandos simples de usar (linguagem de alto nível). Analise as implementações em Python e em C do algoritmo de contagem de ocorrências de uma palavra numa frase, `ocor_palavras.*`, e os programas `bit_op.*`. Identifique (a) o bloco de instruções em C que corresponde à linha de instrução "`numOcor = frase.count (palavra)`" em Python e (b) o bloco de instruções em C que corresponde à instrução "`bin(c)`" em Python?

5. **Mascaramento de *bits***: O procedimento de setar um subconjunto de *bits* de uma variável em 1 (**mascaramento em 1**), ou resetar um subconjunto de *bits* em 0 (**mascaramento em 0**), mantendo outros *bits* inalterados é fundamental na manipulação dos dados a nível de *bits*. Para isso, usa-se uma **máscara** em que são definidos os *bits* a serem modificados e um operador lógico *bit*-a-*bit*. Quais são a máscara e o operador aplicados no mascaramento em 1 de um subconjunto de *bits* de uma variável? E no mascaramento em 0? Justifique.

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 5 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/3C.html

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