UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

EA871 - Laboratório de programação básica de sistemas digitais

Experimento 1: De C para Python

Nome: Robert Willian Polli RA: 187848

Campinas
Agosto/2022

1. Declaração de variáveis

	Python	C (tipo e tamanho do conteúdo em *byte*)	Entrada/Saída/ de trabalho (Programa em C)
Exercício 13	*n:* int	*n:* int, 4 bytes	Entrada e Saída
	soma: int	*soma:* int, 4 bytes	Trabalho
	divisor: int	*divisor:* int, 4 bytes	Trabalho
pertence.*	*n:* int	*n:* int, 4 bytes	Entrada
	i: int	*i:* int, 4 bytes	Trabalho
	x: float	*x:* float, 4 bytes	Entrada e Saída
	y: float	*y:* float, 4 bytes	Entrada e Saída
ocor_palavras.*	*frase:* string	*ocorre:* char, 1 byte	Trabalho
	palavra: string	*m, n:* int, 4 bytes	Saída
	m: int	*i, j :* int, 4 bytes	Trabalho
	n: int	*numOcor:* int, 4 bytes	Saída
	numOcor: int	*frase:* char, 80 bytes	Entrada e Saída
	- - -	*palavra:* char, 80 bytes	Entrada e Saída

Para se determinar o tamanho das variáveis nos programas em C, alterou-se o código base e adicionou-se comandos de "*sizeof()*", onde foram passados como argumento as variáveis de interesse. Abaixo é mostrado um exemplo para o programa "*exercicio13.c*".

```
printf("Tamanhos: \n n=%ld\n soma=%ld\n divisor=%ld\n", sizeof(n), sizeof(soma), sizeof(divisor));
return 0;
```

Executando os programas, obteve-se saídas como a mostrada abaixo. A unidade para o comando "*sizeof()*" é *byte*.

```
Tamanhos:
 n=4
 soma=4
 divisor=4
```

3. Operadores bit-a-bit

	Nome do Operador	Python	C
a & b	And	20	20
a | b	Or	127	127
a ^ b	Xor	107	107
~a	Not	-88	168
~b	Not	-61	-61
a << 2	Deslocamento à esquerda em dois bits	348	92
a >> 2	Deslocamento em dois bits à direita	21	21

AND, OR, XOR: Todas as operações não consideram o valor do *bit carry*. As lógicas booleanas são aplicadas apenas *bit* a *bit*, em ambas as linguagens.

NOT: No código em C, "~a" e "~b" estão sendo calculado *bit* a *bit*, invertendo-se cada um. Entretanto, o cálculo de "~b" está sendo interpretado como um número negativo, em complemento de dois.

Em python, o operador lógico "~" realiza a operação "~x = - x - 1", ou seja, além de aplicar o complemento de dois sobre o número de entrada, ainda subtrai uma unidade deste.

Deslocamento à esquerda: A diferença ocorreu devido à forma do interpretador apresentar o resultado. No interpretador python, foi considerado o valor do "*carry*" gerado devido ao deslocamento, o que não ocorreu no interpretador em C. Como mostrado abaixo:

Python: 0 1 0 1 0 1 1 1 (2) << 2 = 0 0 0 1 0 1 0 1 1 1 0 0 (2) = 348 (10)

C: 0 1 0 1 0 1 1 1 (2) << 2 = - - - - 0 1 0 1 1 1 0 0 (2) = 92 (10)

4. Nível de abstração

a)

Este primeiro bloco é responsável por contar o número de ocorrência de palavras na frase.

```
// conte a quantidade de ocorrências
for (i=0; i <= m-n; i++) {
        if (frase[i] == palavra[0]) {
           ocorre = 1;
          for (j=1; j<n; j++) {
              if (frase[i+j] != palavra[j]) {
                    ocorre = 0;
                    j++;
                    break;
              }
           }
           numOcor = numOcor+ocorre;
           if (ocorre) i=i+j-1;
    }
```

b)

O bloco de instrução responsável por transformar um número decimal em binário é mostrado a seguir.

Para ficar em conformidade com o código em python, inicialmente é printado ao usuário a base do número “0b”, em seguida é realizada sete operações lógicas *“and” bit-a-bit*.

A operação inicia-se com o bit mais significativo, realizando-se uma and entre o sétimo bit do número “value” com o número binário 1 << 7 = 1000000(2). Seu resultado sofre um deslocamento para a direita para melhor adequar-se na apresentação do resultado ao usuário. Tal ação é repetida para todos os oito bits.

```
void bin(uint8_t value)
  {
    int i;

    printf("0b");
    for (i = 7; i >= 0; i--)
    {
      printf("%d", (value & (1 << i)) >> i );
    }
    return;
}
```

5. Mascaramento de bits

No mascaramento em “1”, pode-se utilizar o operador lógico bit-a-bit “or”. O número a ser utilizado como máscara deve conter “0” nos bits que não serão alterados, como no exemplo abaixo.

Registrador	*0 0 1 0 1 1 0*
Máscara	*0 0 1 1 0 0 0* *(OR)*
Novo valor	*0 0 1 1 1 1 0*

Para o mascaramento em “0”, o operador lógico bit-a-bit utilizado pode ser do tipo “and”. O número a ser utilizado como máscara deve conter “1” nos bits que não serão alterados, como no exemplo abaixo.

Registrador	*0 1 0 0 1 0 1*
Máscara	*1 1 1 1 0 0 1* *(AND)*
Novo valor	*0 1 0 0 0 0 1*