USP - Universidade de São Paulo
Instituto de Ciências Matemáticas e Computação

Introdução à Ciência da Computação I
SCC0221

LISTA DE EXERCÍCIOS III - FUNÇÕES, CADEIAS DE CARACTERES E PONTEIROS

Em todos os exercícios, considere que o usuário pode digitar valores não correspondentes ao que é pedido (por exemplo, dar de entrada um número negativo enquanto o que foi pedido foi um número natural). Recomenda-se que essas exceções sejam tratadas utilizando estruturas condicionais (por exemplo, finalizando a execução do programa previamente através de `exit(0)` ou `return 0`).

Dê preferência em utilizar alocação dinâmica para dados com tamanho significativo (como, por exemplo, cadeias de caracteres e *structs*). Lembre-se de modularizar o código com a possível utilização de funções onde vier a ser conveniente.

Parte I - Funções

1. Qual a principal vantagem de se utilizar funções ao invés de reduzir o código todo a uma só função `main`? Reflita quais critérios você poderia utilizar para determinar se um segmento do código seria melhor estruturado se estivesse em uma função.

2. Agora que você conhece o que são valores de retorno em uma função, reflita qual o significado da expressão `return 0` na função principal de um código em C.

3. É possível que sejam retornados dois valores diferentes em uma função? Se sim, explique como funciona. Se não, existe alguma alternativa?

4. Diferencie passagem de parâmetros por valor de passagem de parâmetros por referência.

5. Suponha que queiramos criar uma função que realize a soma de dois números inteiros armazenados nas variáveis `a` e `b`, gerando um terceiro número inteiro de saída que deve ser armazenado na variável `c`. Na linguagem C, é possível que estruturemos o cabeçalho dessa função de duas formas distintas: na primeira, o retorno da função é o valor resultante da soma; na segunda, o retorno da função é `void`. Defina o cabeçalho dessas duas formas e discuta seu funcionamento.

Parte II - Ponteiros e Alocação Dinâmica

1. Diferencie as principais características entre memória *Stack* de memória *Heap*. Por que é mais aconselhável que dados que ocupem grande quantidade de *bytes* (ou sejam de tamanho variável) sejam armazenados na memória *Heap*?

2. O que é um ponteiro? O que um ponteiro possui de diferente de outros tipos de dados convencionais? Qual a relação de um ponteiro com a memória *Heap*?

3. Você consegue enxergar a relação entre vetores e ponteiros? Qual é ela?

4. Avalie as seguintes funções, encontradas em um código em C. Podemos dizer que elas funcionam como devem? Se não, o que há de errado? Se sim, o que cada uma está fazendo?

Função 1

```
void whatIsThis(int *firstValue, int *secondValue) {
    int temp = firstValue;
    firstValue = secondValue;
    secondValue = temp;
}
```

Função 2

```
void mysteriousFunction(int *firstValue, int *secondValue) {
    int *temp = firstValue;
    firstValue = secondValue;
    secondValue = temp;
}
```

Função 3

```
void omgItWorks(char *newString) {
    newString = (char *) malloc(15);
    char *address = &newString;
    return address;
}
```

Função 4

```
int *pipipiInCuscuz(int n) {
    int *v = (int *) calloc(20, sizeof(int));
    return v;
}
```

5. Diferencie alocação estática de alocação dinâmica. Pontue, também, as diferenças das seguintes declarações de vetores, levando em conta o local de armazenamento, seu conteúdo interno e suas permissões de leitura/escrita:

```
char firstVector[50];
char secondVector[] = "Hello World!";
char *thirdVector = (char *) malloc(50*sizeof(char));
char *fourthVector = "Hello World!";
char *fifthVector = (char *) calloc(50, sizeof(char));
```

6. Escreva um código em C que receba uma cadeia de caracteres composta por exatamente 50 caracteres e exiba na tela, **sem indexar o vetor de caracteres diretamente**, a mesma cadeia de caracteres sem os caracteres das posições pares.
   Dica: Utilize um ponteiro de caractere (`char *`) para percorrer o vetor.

7. Na linguagem C, um número inteiro é comumente representado utilizando 4 *bytes* (ou 32 bits). Contrariando o senso comum, é possível que interpretemos esse inteiro, através da utilização de ponteiros, como quatro caracteres distintos, uma vez que um caractere ocupa, na memória, 1/4 do número de bytes de um inteiro.
   Escreva um algoritmo em C que receba da entrada padrão (`stdin`) um número inteiro e leia esse inteiro como quatro caracteres distintos.

8. Considere os dois cabeçalhos, provenientes de uma mesma função, e uma *struct* definida a seguir e repare na sutil diferença de declaração destes: enquanto um passa a própria estrutura nos parâmetros da função, o outro passa um ponteiro para esta. Pontue as principais vantagens/desvantagens de se utilizar a passagem de um ponteiro de estrutura ao invés da própria estrutura como parâmetro de uma função.

```
typedef struct _regStruct {
    int codeIdentifier;
    char *ownerName;
    int ownerAge;
} RegStruct_t;

// Cabecalho 1
void updateOwnerName(RegStruct_t currRegistry, char *newOwnerName);

// Cabecalho 2
void updateOwnerName(RegStruct_t *currRegistry, char *newOwnerName);
```

Parte III - Cadeias de caracteres (*strings*) e *structs*

1. Considere uma situação em que possui-se dados de clientes de uma empresa em forma de registros, isto é, cada cliente possui um código identificador, um nome e uma idade. De que maneira esses dados poderiam ser armazenados em um programa em C de maneira eficiente?

2. Construa, sem utilizar dos artifícios da biblioteca `string.h`, as funções que realizem as funcionalidades abaixo:

   (i) Uma função que encontre o tamanho de uma cadeia de caracteres dado como parâmetro. A função deve retornar o tamanho como um número inteiro.

   (ii) Uma função que verifique se duas cadeias de caracteres são iguais. A função deve retornar `0` caso sejam, e `1` caso não sejam, e deve ser *case sensitive*.

   (iii) Uma função que cheque se uma dada cadeia de caracteres dada como parâmetro é palíndromo. A função deve retornar `0` em caso afirmativo, e `1` em caso negativo.

   (iv) Uma função que verifique se os n primeiros caracteres de duas cadeias de caracteres dadas como parâmetro são iguais. A função deve retornar `0` caso sejam, e `1` caso não sejam, e deve ser *case sensitive*.

   (v) Uma função que converta todos os caracteres de uma cadeia para letras maiúsculas (*uppercase*). A função deve retornar a nova string com as adaptações corretas.

   (vi) Uma função que converta todos os caracteres de uma cadeia para letras minúsculas (*lowercase*). A função deve retornar a nova string com as adaptações corretas.

3. Construa um programa em C que receba uma lista de nomes simples e, por último, um sobrenome. A saída desse programa deve exibir, na saída padrão, todos os nomes seguidos pelo sobrenome informado de entrada.
   Dica: A função `strcat` da biblioteca `string.h` pode te ajudar.

4. Construa uma função em linguagem C que duplique uma string dada como parâmetro. A função deve retornar uma cópia da string em outro ponteiro.
   Dica: A função `strcpy` da biblioteca `string.h` pode te ajudar.

5. Em computação, é extremamente importante que consigamos, também, ordenar um vetor de strings em ordem alfabética. O algoritmo construído na lista passada, o *Bubble Sort*, pode ser utilizado também para ordenar em ordem alfabética caso sejam feitas algumas adaptações.
   Construa um algoritmo *Bubble Sort* que ordene um vetor de strings por ordem alfabética utilizando linguagem C.
   Dica: A função `strcmp` da biblioteca `string.h` pode te ajudar. Além disso, não é necessário mudar as posições dos caracteres de cada string, basta mudar o endereço de toda a string!

6. Considere a função escrita em linguagem C escrita a seguir, que se relaciona com uma *struct* arbitrária denominada `regStruct_t`:

```c
typedef struct regStruct {
    int code;
    char *name;
    int age;
} regStruct_t;

regStruct_t *createNewRegistry(int codeIdentifier, char *newName, int newAge) {
    regStruct_t *newStruct = (regStruct_t *) malloc(sizeof(regStruct_t));
    if (newStruct != NULL) {
        newStruct.code = codeIdentifier;
        newStruct.name = newName;
        newStruct.age = newAge;
    }
    return newStruct;
}
```

   É possível dizer que a função possui funcionamento correto? Se sim, explique passo a passo o que ela está fazendo. Se não, o que há de errado?

7. Em um hospital, o controle de pacientes é feito através de registros que são carregados em memória, onde cada registro de paciente possui um **código identificador** e informação relativa ao seu **nome**, **idade** e **infecção por coronavírus** (sim/não). Construa, utilizando a linguagem C, um programa que permita a manutenção de registros de pacientes desse hospital onde alguns deles estão infectados com coronavírus. O programa deve ser capaz de:

   (i) Inserir os dados de um novo paciente como um novo registro, com os dados informados de entrada.

   (ii) Remover o registro já existente de algum paciente.

   (iii) Listar todos os pacientes cadastrados.

   (iv) Atualizar alguma das informações relativas a um registro já existente.

   (v) Informar a porcentagem de pacientes que possuem coronavírus.

   (vi) Informar o número total de pacientes.

   Dica: Pode ser utilizado um vetor de *structs* alocado dinamicamente na memória para armazenar os dados.

Bom divertimento! :)