Exercícios de Programação em C

Fundamentos de Programação e Especificação de Protocolos
Professora Cristina Nader Vasconcelos
Monitor Álvaro Fernandes de Abreu Justen
Universidade Federal Fluminense

1. Quais serão os valores de x, y e p ao final do trecho de código abaixo?

```
int x, y, *p;
y = 0;
p = &y; //*p = 0
x = *p; //x = 0
x = 4; //x = 4
(*p)++; //*p = 1, y = 1
--x; //x = 3
(*p) += x; //*p = 4, y = 4
```

Ao final, temos:

x = 3, y = 4, p apontando para y (*p = 4).

2. Os programas (trechos de código) abaixo possuem erros. Qual(is)? Como deveriam ser?

a)

```
void main() {
int x, *p;
x = 100;
p = x; //p deveria receber o endereço de x, já que p é um ponteiro (e x não).
Ponteiros “armazenam” o endereço para o qual eles apontam! O código correto
seria: p = &x;
printf(“Valor de p: %d.\n”, *p);
}
```

b)

```
void troca (int *i, int *j) {
int *temp;
*temp = *i;
*i = *j;
*j = *temp;

}
```

A variável “temp” não precisava ser um ponteiro, já que apenas precisa armazenar um valor inteiro, sem precisar apontar para algum lugar. O código correto seria:

```
void troca (int *i, int *j) {
    int temp;
    temp = *i;
    *i = *j;
    *j = temp;
}
```

c)

```
char *a, *b;
a = "abacate"; //o ponteiro “a” ainda não aponta para algum lugar nem possui memória alocada!
b = "uva"; //o ponteiro “b” ainda não aponta para algum lugar nem possui memória alocada!
if (a < b)
   printf ("%s vem antes de %s no dicionário", a, b);
else
   printf ("%s vem depois de %s no dicionário", a, b);
```

O correto seria:

```
    char a[] = "abacate", b[] = "uva";
    if (a[0] < b[0]) {
        printf ("%s vem antes de %s no dicionário", a, b);
    }
    else {
        printf ("%s vem depois de %s no dicionário", a, b);
    }
```

Nesse caso, verificar apenas a primeira letra das cadeias de caracteres funciona, pois temos “a” e “u”. Porém, o código acima não funcionaria para os casos “manga” e “mamão”, por exemplo. Fica como exercício para o aluno criar uma função que faz a verificação de toda a cadeira de caracteres.

3) Suponha que os elementos do vetor v são do tipo int e cada int ocupa 8 bytes no seu computador. Se o endereço de v[0] é 55000, qual o valor da expressão v + 3?

   Se v (ou o endereço de v[0]), que representa o primeiro item do vetor está no byte de endereço 55000, logo o índice v[3] (ou v + 3) estará no byte 55000 + 8*3 = 55024. Nota: em máquinas de 32 bits, inteiros ocupam 32 bits (4 bytes).

4) Escreva uma função mm que receba um vetor inteiro v[0..n-1] e os endereços de duas variáveis inteiras, digamos min e max, e deposite nessas variáveis o valor de um elemento mínimo e o valor de um elemento máximo do vetor. Escreva também uma função main que use a função mm.

```
#include <stdio.h>
#include <stdlib.h>

void mm(int *v, int n, int *min, int *max) {
   int i;
   *min = v[0];
   *max = v[0];
   for (i = 1; i < n; i++) {
      if (v[i] > *max) {
         *max = v[i];
      }
      else if (v[i] < *min) {
         *min = v[i];
      }
   }
}

int main() {
   int n, i, *vet, minimo, maximo;
   printf("Quantos numeros voce deseja digitar? ");
   scanf("%d", &n);
   vet = malloc(n * sizeof(int));
   for (i = 0; i < n; i++) {
      printf("Digite o numero de indice %d: ", i);
      scanf("%d", &vet[i]);
   }

   mm(vet, n, &minimo, &maximo);
   printf("Minimo: %d. Maximo: %d.\n", minimo, maximo);
   return 0;
}
```

5) Suponha que v é um vetor. Descreva a diferença conceitual entre as expressões v[3] e v + 3.

Como “v” nos retorna o endereço do primeiro elemento de um vetor, “v + 3” nos retorna o endereço do quarto elemento. Porém, v[3] nos retorna o quarto elemento! A diferença é que em um caso temos o elemento e em outro o endereço do elemento.

6) (sem usar o computador) Qual o conteúdo do vetor a depois dos seguintes comandos.

```
int a[99];
for (i = 0; i < 99; ++i)     a[i] = 98 - i;
for (i = 0; i < 99; ++i)     a[i] = a[a[i]];
```

O primeiro laço “for” popula o vetor “a” com números decrescentes, começando de 98 (para o índice 0) até 0 (para o índice 98). Já o segundo, onde temos o vetor populado, troca os valores dos índices; ele é mais complicado: para cada índice “i”, ele troca o valor do elemento no vetor pelo valor do elemento que possui como índice o valor do elemento a[i]. Dessa forma, até a primeira metade do vetor (índice 49, pois 99 / 2 = 49), os valores são invertidos (de 0 a 49, para os índices de 0 a 49). Porém, para os próximos itens, o vetor já está invertido, então ele é sobrescrito com os novos valores (da primeira metade do vetor), que, por coincidência, são os valores que estavam anteriormente (quando populamos o vetor, no primeiro “for”, em ordem decrescente).

7) Escreva uma função chamada troca que troca os valores dos parâmetros recebidos. Sua assinatura deve ser:
void troca(float *a, float *b);

```
void troca (float *a, float *b) {
   float temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
```

8) Crie uma função que receba uma string como parâmetro (de tamanho desconhecido) e retorne uma cópia da mesma. A assinatura da função deve ser:
char *strcopy(char *str);

```
char *strcopy(char *str) {
   int n, i;
```

```
    char *nova;

    //Primeiro vamos contar quantos caracteres a string tem:
    for (n = 0; str[n] != '\0'; n++) {}

    //Agora vamos copiar alocar memoria para a nova string:
    nova = malloc(n * sizeof(char));

    //Com a memoria alocada, podemos copiar:
    for (i = 0; i <= n; i++) {
        nova[i] = str[i];
    }
    //O "=" eh necessario para copiar tambem o \0 final de str

    return nova;
}
```

9) Escreva uma função que recebe como parâmetros um vetor de inteiros v, o número de elementos dele N e ponteiros para variáveis nas quais devem ser armazenados os valores maximo e minimo do vetor. Sua assinatura deve ser: void maximoMinimo(int *v, int N, int *maximo, int *minimo);

Idêntica à questão 4, trocando apenas “maximoMinimo” por “mm” (e sem necessidade da função main). Nota: na questão 4 não está explícito que precisamos passar como parâmetro o número de elementos do vetor (mas é necessário).

10) Qual o resultado do código abaixo? Explique cada linha.

```
int x = 100, *p, **pp;
p = &x;
pp = &p;
printf(“Valor de pp: %d\n”, **pp);
```

int x = 100, *p, **pp; //x recebe o valor 100, p é um ponteiro para inteiro e pp é um ponteiro para ponteiro para inteiro
p = &x; //p passa a apontar para o endereço de x (logo, *p tem o valor 100)
pp = &p; //pp passa a apontar para o endereço de p, logo *pp é o valor de pp, que é o mesmo que p e **pp é o mesmo que *p, que é o mesmo que x (que é igual a 100)
printf(“Valor de pp: %d\n”, **pp); //imprime o valor de **pp, que é igual ao valor de x, como mencionado na linha acima

11) Escreva uma função que recebe uma string de caracteres e uma letra e devolve um vetor de inteiros contendo as posições (índices no vetor da string) onde a letra foi encontrada) e um inteiro contendo o tamanho do vetor criado (total de letras iguais encontradas). Utilize o retorno de um vetor para retornar os índices e um ponteiro para guardar o tamanho do vetor.

```
#include <stdio.h>
#include <stdlib.h>

int *acha_caractere(char *str, char c, int *pn) {
    int i = 0, n = 0, *indices = 0;

    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == c) {
            n++;
        }
    }
    indices = (int *) malloc(n* sizeof(int));

    for (i = 0, n = 0; str[i] != '\0'; i++) {
        if (str[i] == c) {
            indices[n] = i;
            n++;
        }
    }
    *pn = n;
    return indices;
}


int main() {
    int *indices = 0, n = 0, i;
    char *frase = "teste";

    indices = acha_caractere(frase, 'e', &n);
    for (i = 0; i < n; i++) {
        printf("%d ", indices[i]);
    }
    return 0;
}
```

12) Escreva uma função que codifica uma string em um código secreto. A regra secreta de codificação é extremamente simples: substitui cada letra

pela letra seguinte (Z é codificado como A). Por exemplo, “Estruturas de Dados” se transformaria em “Ftusvuvsbt ef Ebept”. Escreva uma função para codificar e uma para decodificar cadeias segundo este código. Suas funções devem escrever a string produzida em uma string diferente da fornecida como entrada.

```
int strsize(char *str) { //Funcao auxiliar que retorna tamanho de uma string
    int n;
    for (n = 0; str[n] != '\0'; n++) {}
    return n;
}

char *codifica(char *str) {
    int i, tamanho;
    tamanho = strsize(str);
    char *nova = malloc(tamanho * sizeof(int));
    for (i = 0; i < tamanho; i++) {
        if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')) {
            if (str[i] == 'z') {
                nova[i] = 'a';
            }
            else if (str[i] == 'Z') {
                nova[i] = 'A';
            }
            else {
                nova[i] = str[i] + 1;
            }
        }
        else {
            nova[i] = str[i];
        }

    }
    nova[i] = '\0';
    return nova;
}

char *decodifica(char *str) {
    int i, tamanho;
    tamanho = strsize(str);
    char *nova = malloc(tamanho * sizeof(int));
    for (i = 0; i < tamanho; i++) {
```

```
        if ((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z')) {
            if (str[i] == 'a') {
                nova[i] = 'z';
            }
            else if (str[i] == 'A') {
                nova[i] = 'Z';
            }
            else {
                nova[i] = str[i] - 1;
            }
        }
        else {
            nova[i] = str[i];
        }

    }
    nova[i] = '\0';
    return nova;
}
```

13) Escrever um programa que lê duas cadeias `s1` e `s2`, e retorna uma nova cadeia `s3` que contém todos os caracteres que aparecem em `s1` e em `s2`.

```
char *caracteres_repetidos(char *s1, char *s2) {
    int i, j, w, n = 0, encontrado;
    char *s3;

    //Primeiro vamos contar os caracteres repetidos para saber quantos sao:
    for (i = 0; s1[i] != '\0'; i++) {
        for (j = 0; s2[j] != '\0'; j++) {
            if (s1[i] == s2[j]) {
                n++;
                break;
            }
        }
    }
    //Agora podemos alocar memoria para eles:
```

```
    //Alocaremos n + 1 por causa do \0 no final!
    s3 = malloc((n + 1) * sizeof(char));
    n = 0;
    //Vamos buscar novamente os caracteres repetidos:
    for (i = 0; s1[i] != '\0'; i++) {
        for (j = 0; s2[j] != '\0'; j++) {
            if (s1[i] == s2[j]) { //Caractere encontrado nas duas strings!
                //Vamos verificar se o caractere encontrado jah nao foi inserido em s3:
                encontrado = 0;
                for (w = 0; w < n; w++) {
                    if (s3[w] == s1[i]) {
                        encontrado = 1;
                        break;
                    }
                }
                if (encontrado == 0) { //Se nao foi encontrado, adiciona:
                    s3[n] = s1[i];
                    n++;
                    break;
                }
            }
        }
    }
    s3[n] = '\0';

    return s3;
}
```