

**Alunos:** Yuri Victor de Oliveira e Silva e Vinicius de Almeida Gonçalves  
**Matrícula:** 2412130171 ----- 2412130036

## 1. A respeito de funções e procedimentos:

### a) O que diferencia função de procedimento?

R.a) A principal diferença entre uma função e um procedimento é que a função sempre retorna um valor (como int, float, double, entre outros tipos), já o procedimento não retorna valor algum, sendo definido como void. Além disso, uma função normalmente recebe e processa parâmetros de entrada, já o procedimento pode ou não receber algum parâmetro, mas ele nunca devolve um resultado. Pode-se dizer que toda função também é um procedimento, mas nem todo procedimento é uma função, porque os procedimentos não retornam valores.

### b) Procedimentos não podem conter a palavra reservada return? Justifique a sua resposta.

R.b) Procedimentos podem sim conter a palavra reservada return, porém, diferentemente da função que ao usar a palavra reservada return retorna um valor, no procedimento, a palavra é usada para encerrar a execução do próprio procedimento, retornando ao ponto de chamada.

### c) Considerando a operação de inverter o sinal de um número, apresente um exemplo se essa operação fosse apresentada em:

#### i. Uma função

#### ii. Um procedimento

```
1  #include <stdio.h>
2  int fnc_inverter_sinal(int numero) {
3      return -numero;
4  }
5
6  void procedimento_inverter_sinal(int numero) {
7      printf("Número do procedimento depois da inversão: %d\n", -numero);
8  }
9
10
11 int main() {
12     int numero_fnc = -5;
13     printf("Número da função antes da inversão: %d\n", numero_fnc);
14     numero_fnc = fnc_inverter_sinal(numero_fnc);
15     printf("Número da função depois da inversão: %d\n", numero_fnc);
16
17     int numero_procedimento = -10;
18     printf("Número do procedimento antes da inversão: %d\n", numero_procedimento);
19     procedimento_inverter_sinal(numero_procedimento);
20
21     return 0;
22 }
```

### d) Qual é a diferença entre passagem de parâmetros por valor e por referência? Dê um exemplo em forma de função de cada.

R.d) A diferença de passagem de parâmetros por valor e por referência é como o argumento é tratado dentro da função. Na passagem por valor, é passado apenas uma cópia do valor da variável para a função. Já na passagem por referência, é passado o endereço da variável através de um ponteiro. Dessa forma, as alterações feitas na função também alteram o valor da variável original.

Exemplo da letra d):

```
1  #include <stdio.h>
2  void numeroX2_valor(int numero) {
3      numero = numero * 2;
4      printf("Passagem por valor: %d\n", numero);
5  }
6
7  void numeroX2_referencia(int *numero) {
8      *numero = *numero * 2;
9      printf("Passagem por referência: %d\n", *numero);
10 }
11
12 int main() {
13
14     int numero_valor = 5;
15     printf("Valor original (valor): %d\n", numero_valor);
16     numeroX2_valor(numero_valor);
17     printf("Número na main depois de chamar a função (valor): %d\n", numero_valor);
18
19     int numero_referencia = 10;
20     printf("Valor original (referência): %d\n", numero_referencia);
21     numeroX2_referencia(&numero_referencia);
22     printf("Número na main depois de chamar a função (referência): %d\n", numero_referencia);
23
24     return 0;
25 }
```

2. Escreva as seguintes funções (ou procedimentos, caso seja um procedimento).

Para esta questão considere enviar apenas a função ou o procedimento, não é relevante o envio do desenvolvimento da main. Este desenvolvimento só será útil para você testar se sua função tem o comportamento esperado.

a) int abs(int x) - Devolve o valor absoluto de x.

i. abs(-5): retorna 5

ii. abs(5): retorna 5

```
5  // 2.a)
6  int abs(int x) {
7      x *= -1;
8  }
9
```

b) int eVogal(char ch) - Verifica se ch é uma das vogais do alfabeto (minúscula ou maiúscula).

i. eVogal('o'): retorna 1 (true).

ii. eVogal('L'): retorna 0 (false)

```
10 // 2.b)
11 int eVogal(char ch) {
12     char vogais[] = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'};
13     for (int i = 0; i < 10; i++) {
14         if (ch == vogais[i]) {
15             return 1;
16         }
17     }
18     return 0;
19 }
```

c) `int eQuadrado(x, y)` - Devolve um valor lógico que indica se `x` é ou não igual a  $y^2$ .

i. `eQuadrado(9, 3)`: retorna 1 (true).

ii. `eQuadrado(3, 9)`: retorna 0 (false).

```
21 // 2.c)
22 int eQuadrado(int x, int y) {
23     if (x == y*y) {
24         return 1;
25     }
26     else {
27         return 0;
28     }
29 }
```

d) `double obterSegundos(double horas)` - Devolve o número de segundos existentes em um conjunto de horas.

i. `obterSegundos(0)`: retorna 0.

ii. `obterSegundos(1)`: retorna 3600.

iii. `obterSegundos(2)`: retorna 7200.

```
31 // 2.d)
32 double obterSegundos(double horas) {
33     double segundos = horas * 60 * 60;
34     return segundos;
35 }
36
```

e) `double converterHoras(double horas, char formato)` - Semelhante a função anterior, só que recebe mais um parâmetro indicando o formato desejado que se quer converter, sendo: 'h' ou 'H' - Horas, 'm' ou 'M' - Minutos e 's' ou 'S' - Segundos. Caso for especificado um formato inválido, retorna -1.

i. `converterHoras(3, 'H')`: retorna 3.

ii. `converterHoras(3, 'm')`: retorna 180.

iii. `converterHoras(3, 'S')`: retorna 10800.

iv. `converterHoras(3, 'x')`: retorna -1.

```
37 // 2.e)
38 double converterHoras(double horas, char formato) {
39     double segundos = horas * 60 * 60;
40     double minutos = horas * 60;
41     if (formato == 's' || formato == 'S') {
42         return segundos;
43     }
44     else if (formato == 'm' || formato == 'M') {
45         return minutos;
46     }
47     else if (formato == 'h' || formato == 'H') {
48         return horas;
49     }
50     else {
51         return -1;
52     }
53 }
54
```

f) void `exibirMedia(int n, int *vet)` - Que calcula a média de um vetor de `n` números e a exibe para o usuário.

i. `int vet[] = {5, 7, 9}; exibirMedia(vet);` informa ao usuário: “média: 7”.

```
55 // 2.f)
56 void exibirMedia(int n, int *vet) {
57     double soma = 0;
58     for (int i = 0; i < n; i++) {
59         soma += vet[i];
60     }
61     double media = soma / n;
62     printf("Média: %.1f\n", media);
63 }
64
```

g) `int buscarValor(int n, int *vetor, int busca)` - Que recebe como parâmetro um vetor de `n` inteiros e o valor a ser buscado dentro do vetor. A função deverá retornar o índice do vetor, onde o valor foi encontrado. Caso o índice não for encontrado, a função retorna -1, um índice inválido no vetor.

```
65 // 2.g)
66 int buscarValor(int n, int *vetor, int busca) {
67     for (int i = 0; i < n; i++) {
68         if (busca == vetor[i]) {
69             return i;
70         }
71     }
72     return -1;
73 }
```

h) void `potencia(int *x, int y)` - Que obtém um endereço de memória de um inteiro e o eleva em potência de `y` dentro da função.

i. `int n = 5; potencia(&n, 2);` → a variável `n` deverá ter o resultado 25 ( $5^2$ ).

```
76 // 2.h)
77 void potencia(int *x, int y) {
78     printf("O resultado de %d elevado a %d é: %.1f\n", *x, y, pow(*x, y));
79 }
```

i) `int preencherMatriz(int m, int n, int matriz[m][n])` - Que recebe uma matriz `mXn` e preenche cada um de seus índices com um novo valor obtido do usuário.

```
81 // 2.i)
82 int preencherMatriz(int m, int n, int matriz[m][n]) {
83     for (int i = 0; i < m; i++) {
84         for (int j = 0; j < n; j++) {
85             printf("Digite o valor para matriz[%d][%d]: ", i, j);
86             scanf("%d", &matriz[i][j]);
87         }
88     }
89     return 0;
90 }
```

j) Escreva um procedimento em linguagem C, capaz de realizar a troca de dois números passados como parâmetro, dentro do procedimento. Isto é, após a execução do procedimento, as variáveis, cujos endereços foram passados ao procedimento, deverão ter os valores trocados entre si.

```
92 // 2.j)
93 void trocar_numeros(int *a, int *b) {
94     int valor_a = *a;
95     *a = *b;
96     *b = valor_a;
97 }
```

3. Utilizando alocação dinâmica, faça um programa que implemente as seguintes funções, cada qual deve retornar um bloco de memória alocado dinamicamente na Heap.

a) int \*build\_int\_array(unsigned int size), retorna um array de inteiros alocado dinamicamente na Heap de tamanho size.

```
4 // 3.a)
5 int *build_int_array(unsigned int size) {
6     int *array = (int *)malloc(size * sizeof(int));
7     if (array == NULL) {
8         printf("Erro ao alocar memória.\n");
9         return NULL;
10    }
11    for (unsigned int i = 0; i < size; i++) {
12        array[i] = 0;
13    }
14    return array;
15 }
```

```
21 // 3.a) *(main)*
22 unsigned int size;
23 printf("Digite o tamanho do array: ");
24 scanf("%u", &size);
25 int *meuArray = build_int_array(size);
26 if (meuArray != NULL) {
27     for(unsigned int i = 0; i < size; i++) {
28         printf("Elemento %u: %d\n", i, meuArray[i]);
29     }
30     free(meuArray);
31 }
```

b) `float *medias_das_notas(int m, int n, float matriz[m][n])`, onde `m` é o número de alunos, `n` é o número de notas de cada aluno, `matriz` relaciona os alunos com suas respectivas notas, e `*medias_das_notas` retorna um array unidimensional da média de notas de cada aluno. Exemplo:

Entrada da função:

```
char m = 3, n = 2, matriz[3][2] = {
{5, 7}, // aluno do índice 0, notas 5 e 7
{8, 7}, // aluno do índice 0, notas 8 e 7
{6, 10}, // aluno do índice 0, notas 6 e 10
};
```

Saída da função:

(float \*) {7, 7.5, 8} (um vetor das médias dos alunos, alocado dinamicamente na memória heap). Saída da função: (char \*) "AnaAnaAna" (memória heap: {'A', 'n', 'a', 'A', 'n', 'a', 'A', 'n', 'a', '\0'})

```
17 // 3.b)
18 float *medias_das_notas(int m, int n, float matriz[m][n]) {
19     float *medias = (float *)malloc(m * sizeof(float));
20     if (medias == NULL) {
21         printf("Erro ao alocar memória para as médias.\n");
22         return NULL;
23     }
24     for (int i = 0; i < m; i++) {
25         float soma = 0;
26         for (int j = 0; j < n; j++) {
27             soma += matriz[i][j];
28         }
29         medias[i] = soma / n;
30     }
31     return medias;
32 }
```

```
50 // 3.b) *(main)*
51 int m = 3;
52 int n = 2;
53 float matriz[3][2] = {
54     {5, 7},
55     {8, 7},
56     {6, 10}
57 };
58 float *resultado = medias_das_notas(m, n, matriz);
59 if (resultado != NULL) {
60     for (int i = 0; i < m; i++) {
61         printf("Média do aluno %d: %.1f\n", i, resultado[i]);
62     }
63     free(resultado);
64 }
```

c) `char *repete_string(char str[], int n)`, retorna uma nova string alocada dinamicamente com `n` repetições de `str`.

Exemplo:

Entrada da função:

`char str[] = "Ana"` (memória stack: {'A', 'n', 'a', '\0'}), `n = 3`.

Saída da função:

(`char *`) "AnaAnaAna" (memória heap: {'A', 'n', 'a', 'A', 'n', 'a', 'A', 'n', 'a', '\0'}).

```
35 // 3.c)
36 char *repete_string(char str[], int n) {
37     int tamanho_orig = strlen(str);
38     int tamanho_nova = tamanho_orig * n;
39     char *nova_string = (char *)malloc((tamanho_nova + 1) * sizeof(char));
40     if (nova_string == NULL) {
41         printf("Erro ao alocar memória para a nova string.\n");
42         return NULL;
43     }
44     for (int i = 0; i < n; i++) {
45         strcpy(nova_string + (i * tamanho_orig), str);
46     }
47     return nova_string;
48 }
```

```
80 // 3.c) *(main)*
81 char str[] = "Ana";
82 int n = 3;
83 char *resultado = repete_string(str, n);
84 if (resultado != NULL) {
85     printf("Resultado: %s\n", resultado);
86     free(resultado);
87 }
```