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

```c
#include <stdio.h>
int fnc_inverter_sinal(int numero) {
    return -numero;
}

void procedimento_inverter_sinal(int numero) {
    printf("Número do procedimento depois da inversão: %d\n", -numero);
}


int main() {
    int numero_fnc = -5;
    printf("Número da função antes da inversão: %d\n", numero_fnc);
    numero_fnc = fnc_inverter_sinal(numero_fnc);
    printf("Número da função depois da inversão: %d\n", numero_fnc);

    int numero_procedimento = -10;
    printf("Número do procedimento antes da inversão: %d\n", numero_procedimento);
    procedimento_inverter_sinal(numero_procedimento);

    return 0;
}
```

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):

```c
#include <stdio.h>
void numeroX2_valor(int numero) {
    numero = numero * 2;
    printf("Passagem por valor: %d\n", numero);
}

void numeroX2_referencia(int *numero) {
    *numero = *numero * 2;
    printf("Passagem por referência: %d\n", *numero);
}

int main() {

    int numero_valor = 5;
    printf("Valor original (valor): %d\n", numero_valor);
    numeroX2_valor(numero_valor);
    printf("Número na main depois de chamar a função (valor): %d\n", numero_valor);

    int numero_referencia = 10;
    printf("Valor original (referência): %d\n", numero_referencia);
    numeroX2_referencia(&numero_referencia);
    printf("Número na main depois de chamar a função (referência): %d\n", numero_referencia);

    return 0;
}
```

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 util 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

```c
// 2.a)
int abs(int x) {
    x *= -1;
}
```

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)

```c
// 2.b)
int eVogal(char ch) {
    char vogais[] = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'};
    for (int i = 0; i < 10; i++) {
        if (ch == vogais[i]) {
            return 1;
        }
    }
    return 0;
}
```

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).

```c
// 2.c)
int eQuadrado(int x, int y) {
    if (x == y*y) {
        return 1;
    }
    else {
        return 0;
    }
}
```

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.

```c
// 2.d)
double obterSegundos(double horas) {
    double segundos = horas * 60 * 60;
    return segundos;
}

```

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.

```c
// 2.e)
double converterHoras(double horas, char formato) {
    double segundos = horas * 60 * 60;
    double minutos = horas * 60;
    if (formato == 's' || formato == 'S') {
        return segundos;
    }
    else if (formato == 'm' || formato == 'M') {
        return minutos;
    }
    else if (formato == 'h' || formato == 'H') {
        return horas;
    }
    else {
        return -1;
    }
}

```

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”.

```c
// 2.f)
void exibirMedia(int n, int *vet) {
    double soma = 0;
    for (int i = 0; i < n; i++) {
        soma += vet[i];
    }
    double media = soma / n;
    printf("Média: %.1f\n", media);
}

```

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.

```c
// 2.g)
int buscarValor(int n, int *vetor, int busca) {
    for (int i = 0; i < n; i++) {
        if (busca == vetor[i]) {
            return i;
        }
    }
    return -1;
}
```

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).

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

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

```c
// 2.i)
int preencherMatriz(int m, int n, int matriz[m][n]) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("Digite o valor para matriz[%d][%d]: ", i, j);
            scanf("%d", &matriz[i][j]);
        }
    }
    return 0;
}
```

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.

```c
// 2.j)
void trocar_numeros(int *a, int *b) {
    int valor_a = *a;
    *a = *b;
    *b = valor_a;
}
```

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.

```c
// 3.a)
int *build_int_array(unsigned int size) {
    int *array = (int *)malloc(size * sizeof(int));
    if (array == NULL) {
        printf("Erro ao alocar memória.\n");
        return NULL;
    }
    for (unsigned int i = 0; i < size; i++) {
        array[i] = 0;
    }
    return array;
}
```

```c
    // 3.a) *(main)*
    unsigned int size;
    printf("Digite o tamanho do array: ");
    scanf("%u", &size);
    int *meuArray = build_int_array(size);
    if (meuArray != NULL) {
        for(unsigned int i = 0; i < size; i++) {
            printf("Elemento %u: %d\n", i, meuArray[i]);
        }
        free(meuArray);
    }
```

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 dinâmicamente na memória heap). Saída da função: (char *) “AnaAnaAna” (memória heap: {'A', 'n', 'a', 'A', 'n', 'a', 'A', 'n', 'a', '\0'})

```c
// 3.b)
float *medias_das_notas(int m, int n, float matriz[m][n]) {
    float *medias = (float *)malloc(m * sizeof(float));
    if (medias == NULL) {
        printf("Erro ao alocaar memória para as médias.\n");
        return NULL;
    }
    for (int i = 0; i < m; i++) {
        float soma = 0;
        for (int j = 0; j < n; j++) {
            soma += matriz[i][j];
        }
        medias[i] = soma / n;
    }
    return medias;
}
```

```c
    // 3.b) *(main)*
    int m = 3;
    int n = 2;
    float matriz[3][2] = {
        {5, 7},
        {8, 7},
        {6, 10}
    };
    float *resultado = medias_das_notas(m, n, matriz);
    if (resultado != NULL) {
        for (int i = 0; i< m; i++) {
            printf("Média do aluno %d: %.1f\n",i , resultado[i]);
        }
        free(resultado);
    }
```

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'}).

```c
// 3.c)
char *repete_string(char str[], int n) {
    int tamanho_orig = strlen(str);
    int tamanho_nova = tamanho_orig * n;
    char *nova_string = (char *)malloc((tamanho_nova + 1) * sizeof(char));
    if (nova_string == NULL) {
        printf("Erro ao alocar memória para a nova string.\n");
        return NULL;
    }
    for (int i = 0; i < n; i++) {
        strcpy(nova_string + (i * tamanho_orig), str);
    }
    return nova_string;
}
```

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