Pontifícia Universidade Católica de Minas Gerais
Algoritmos e Estruturas de Dados 1 – Engenharia de Software – Campus Lourdes
Profa. Michelle Nery Nascimento
Nome: _____ Nota: _____

Segunda Avaliação – Instruções

1. Leia com bastante atenção o enunciado antes de responder cada questão. A interpretação faz parte da avaliação. A professora não irá responder a nenhuma pergunta durante o período da prova.
2. Esta avaliação é individual e sem consulta. Qualquer semelhança, em parte ou no todo, com respostas de colegas ou de QUALQUER FONTE DE CONSULTA, a nota será zero na prova. Desligue e guarde todos os equipamentos eletrônicos (celular/*smartphone/smartwatch, tablet, laptop, console, fones de ouvido, etc*).
3. Deve ser feita, de preferência, a caneta. Caso use lápis, não cabe recurso de revisão após a prova ser entregue.
4. Duração: 100 minutos. Valor: 25 pontos. Data: 11/11/2024

Questão 1) (5 pontos) Considerando o código abaixo, mostre o que será impresso na tela após a execução do programa.

```
#include <stdio.h>
int a = 3;
float b = 1;

int func3(float a, float *m){
   b += (*m) + 1;
   (*m) += 3;
   printf("(func3) a=%.2f b=%.2f m=%.2f \n",(float)a, (float)b, (float)(*m));
   return a + b;
 }

void func2(int b, float c, float *m){
   b = func3(a, &c);
   (*m) = a + b;
   printf("(func2) a=%.2f b=%.2f c=%.2f m=%.2f\n",(float)a, (float)b,
(float)c,(float)(*m));
 }

int main(){
   int a = 10;
   float res;
   func2(a, b, &res);
   printf("(main) a=%.2f b=%.2f res=%.2f \n",(float)a,(float)b,(float)res);
   return 0;
 }
```

Solução:

Variáveis Globais e Locais

- O código possui uma variável global int a = 3 e float b = 1, e uma variável local int a = 10 definida dentro de main.
- No início da execução, a variável global b = 1 e a variável local a = 10 dentro de main são as mais relevantes para a execução.

Esta função recebe um valor a e um ponteiro m (onde m é uma referência a uma variável float). A função faz os seguintes passos:

1. **b += (*m) + 1;**: Aqui, b (a variável global) é alterada. A variável *m é o valor para o qual o ponteiro m aponta, e ele será modificado na função func2.
2. **(*m) += 3;**: O valor de *m é incrementado em 3.

A impressão mostrará o valor de a, b e *m após essas modificações.

A função func2 é chamada com os seguintes parâmetros:

- b = a (onde a = 10 no main, que será passado para b)
- c = 1 (que é o valor global de b, pois a variável b global tem o valor 1)
- m = &res (ponteiro para res)

Na função func2:

1. **b = func3(a, &c);**: Aqui, a função func3 é chamada com a (global, 3) e o endereço de c. A função func3 faz alterações na variável global b e na variável c, e retorna um valor que é atribuído a b.
2. **(*m) = a + b;**: Depois que func3 retorna, *m (que é res) é atualizado com a soma de a (local, 10) e b (o valor alterado dentro de func3).

A impressão de func2 vai mostrar os valores de a, b, c e *m após essas modificações.

Na função main:

- A variável local a = 10 é passada para func2.
- A variável global b = 1 é passada como c para func2.
- O endereço de res é passado para func2.

Após a execução de func2, o valor de res (modificado dentro de func2 e func3) será impresso.

Passo a Passo da Execução

1. **No main:**
    - a = 10
    - b = 1 (global)
    - res não foi inicializada.
2. **Dentro de func2:**
    - A função func3(a, &c) é chamada com a = 3 (global) e &c (onde c = 1 no início).
    - Dentro de func3:
        - **b += (*m) + 1;**: Aqui, b (global) é modificado: b = 1 + 1 + 1 = 3.
        - **(*m) += 3;**: *m (referência a c) é modificado: c = 1 + 3 = 4.
        - O printf de func3 imprime:
          (func3) a=3.00 b=3.00 m=4.00
        - func3 retorna a + b = 3 + 3 = 6.
    - Depois, dentro de func2, (*m) = a + b = 10 + 6 = 16, ou seja, res = 16.
    - O printf de func2 imprime:
      (func2) a=3.00 b=6.00 c=4.00 m=16.00
3. **De volta ao main:**
    - O valor de a (local) permanece 10.
    - O valor de b (global) foi alterado para 3.
    - O valor de res foi alterado para 16.
    - O printf de main imprime:

(main) a=10.00 b=3.00 res=16.00

Saída final:
(func3) a=3.00 b=3.00 m=4.00
(func2) a=3.00 b=6.00 c=4.00 m=16.00
(main) a=10.00 b=3.00 res=16.00

Apresente um programa em linguagem de Programação C para cada um dos **4 (quatro)** problemas propostos a seguir. A correção irá considerar:

- o atendimento ao problema proposto;
- a qualidade da solução lógica;
- a codificação do programa e suas bibliotecas;
- a endentação (alinhamento) do código e comentários pontuais nos algoritmos.
- a escolha adequada da estrutura de repetição e recursos de modularização;

Questão 2) (5 pontos) Escreva um procedimento que calcule e imprima o valor da série:

$$\P = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \ldots +/- \frac{4}{N}$$

O valor de N será lido do teclado e deve ser positivo. Valide esta condição (o programa só termina a leitura e inicia a execução quando um N válido for lido).

Solução:

```
float Serie(float N){
  float S = 0;
  int sinal = 1;
  for(float i = 1; i <= N; i+=2){
    S+= (sinal)* 4/i;
    sinal *= -1;
  }
  return S;
}
```

```
//Não precisava do main( )

int main(){
  float N;
  printf("Digite um número: ");
  scanf("%f", &N);
  printf("S = %.2f", Serie(N));
  return 0;
}
```

Questão 3) (5 pontos) Escreva uma função **RECURSIVA** que, dado um número positivo e inteiro N, retorne a soma de seus dígitos. **Nesta questão não é necessário implementar o Main( ).**

Exemplo: para o número 9875, a soma será 9 + 8 + 7 + 5 = 29

Solução:

```
int Soma(int N){
  if(N == 0){
    return 0;
  }
  else{
    return N%10 + Soma(N/10);
  }
}
```

```
//Não precisava do main()
int main(){
  int N;
  printf("Digite um número: ");
  scanf("%i", &N);
  printf("Soma = %i", Soma(N));
  return 0;
}
```

Questão 4) (5 pontos) Escreva uma função em C que recebe como parâmetro um vetor de números inteiros e rearranja seus elementos de modo que todos os números negativos venham antes dos números positivos, sem alterar a ordem relativa dos negativos e dos positivos. Retorne o novo vetor. **Nesta questão não é necessário implementar o Main( ).**
Caso sua solução possua mais que o vetor como parâmetro da função, indique os valores iniciais destes.

Regras:

- O vetor pode conter números positivos, negativos e zeros;
- A ordem original dos números negativos e positivos deve ser mantida após o rearranjo;
- O vetor deve ser modificado **in-place** (no mesmo vetor de entrada, sem criar um vetor auxiliar).

Exemplo:
Se o vetor de entrada for: {1, -2, 3, -4, -5, 6}
O vetor resultante deverá ser: {-2, -4, -5, 1, 3, 6}

Solução:

```
void Rearranja(int vetor[], int tamanho) {
int i, j = 0;
   // Procura negativos e move para o início
for (i = 0; i < tamanho; i++) {
      if (vetor[i] < 0) {
         if (i != j) {
            int temp = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = temp;
         }
        j++;
      }
   }
}
```

```
//Não precisava do main

int main() {
   int vetor[] = {1, -2, 3, -4, -5, 6, 0};
   int tamanho = sizeof(vetor) / sizeof(vetor[0]);
   Rearranja(vetor, tamanho);
   printf("Vetor rearranjado: ");
   for (int i = 0; i < tamanho; i++) {
      printf("%d ", vetor[i]);
   }
   printf("\n");
   return 0;
}
```

Questão 5) (5 pontos) O programa a seguir tem por objetivo calcular um recorte de uma matriz passada por parâmetro, e mostrar, ao final, a matriz original e a matriz recortada. Considere que um recorte da matriz é obtido retirando-se as suas bordas, isto é, a primeira e a última linhas e a primeira e a última colunas.

```
void recorta (int mat[TAM][TAM]){
        int recorte[ ? , ? ];
        //complete com seu código
}
```

Abaixo temos um exemplo de matriz (5 X 5) e a matriz recortada correspondente:

$$\begin{bmatrix} 5 & 7 & 4 & 3 & 1 \\ 6 & 7 & 3 & 1 & 4 \\ 0 & 2 & 9 & 8 & 6 \\ 1 & 7 & 1 & 0 & 3 \\ 2 & 6 & 0 & 8 & 2 \end{bmatrix} \quad \begin{bmatrix} 7 & 3 & 1 \\ 2 & 9 & 8 \\ 7 & 1 & 0 \end{bmatrix}$$

Escreva um programa necessário, em C que calcule e mostre o recorte da matriz de acordo com o procedimento acima. Indique, na sua prova, qual o tamanho da matriz original.

Solução:

```
void recorta(int mat[TAM][TAM]) {
   int recorte[TAM-2][TAM-2]; // Definindo a matriz recortada com dimensões reduzidas

   // Copiando os valores da matriz original, sem as bordas, para a matriz recortada
   for (int i = 1; i < TAM - 1; i++) {
      for (int j = 1; j < TAM - 1; j++) {
         recorte[i - 1][j - 1] = mat[i][j];
      }
   }

   printf("Matriz Original:\n");
   for (int i = 0; i < TAM; i++) {
      for (int j = 0; j < TAM; j++) {
         printf("%d ", mat[i][j]);
      }
      printf("\n");
   }

   printf("\nMatriz Recortada:\n");
   for (int i = 0; i < TAM - 2; i++) {
      for (int j = 0; j < TAM - 2; j++) {
         printf("%d ", recorte[i][j]);
      }
      printf("\n");
   }
}

int main() {
   int mat[TAM][TAM] = { {5, 7, 4, 3, 1}, {6, 7, 3, 1, 4},  {0, 2, 9, 8, 6}, {1, 7, 1, 0, 3},  {2, 6, 0, 8, 2}  };
   recorta(mat);
   return 0;
}
```