Lista 1 (todas os exercícios devem estar implementados em linguagem C)

Nome: Ravi Antônio Gonçalves de Assis
Disciplina: Laboratório de Computação II
Curso: Engenharia de Software

1 – Faça uma função recursiva que retorne o elemento na enésima posição da série de Fibonacci.

```
/*fiboRec: retorna o elemento da enésima posição da série de
fibonacci - Usa recursividade*/
int fiboRec (int n) {
      if (n == 0) {
            return 0;
      }
      return (n == 1 || n == 2) ? 1 : fiboRec(n-1) + fiboRec(n-2);
}
```

2 – Faça uma função iterativa que retorne o elemento na enésima posição da série de Fibonacci.

```
/*fibo: retorna o elemento da enésima posição da série de
fibonacci*/
int fibo(int n) {
      int n3, n1=1, n2=1;
      int i;
      if (n == 1 || n == 2) {
            return n1;
      }
      else if (n == 0) {
            return 0;
      }
      else {
            for (i = 3; i <= n; i++) {
                  n3 = n2 + n1;
                  n1 = n2;
                  n2 = n3;
            }
            return n3;
      }
}
```

3 – Faça uma função recursiva que retorne a soma dos n primeiros termos da série de Fibonacci.

```
/*SomaFiboRec: Retorna a soma dos n primeiros termos da série de
fibonacci - Usa recursividade*/
int SomaFiboRec (int n) {
      if (n <= 0) {
            return 0;
      }
      else {
            return fiboRec(n) + SomaFiboRec(n-1);
      }
}
```

4 – Faça uma função iterativa que retorne a soma dos n primeiros termos da série de Fibonacci.

```
/*SomaFibo: Retorna a soma dos n primeiros termos da série de
fibonacci*/
int SomaFibo (int n) {
      int soma= 0;
      int i;
      for (i=0; i<=n; i++) {
            soma += fibo(i);
      }
      return soma;
}
```

5 – Faça um procedimento recursivo que imprima os elementos de um vetor.

```
/*PrintArrayRec: Imprime os elementos de um vetor - Usa
recursividade*/
void PrintArrayRec (int vet[], int tam, int pos) {
      if (pos < 0 || pos >=tam) {
            return ;
      }
      else {
            printf("[%d]", vet[pos]);
            PrintArrayRec(vet,tam,pos+1);
      }
}
```

6 – Faça um procedimento iterativo que imprima os elementos de um vetor.

```
/*PrintArray: Imprime os elementos de um vetor*/
void PrintArray(int vet[], int tam) {
	int i;
	for (i = 0; i < tam; i++) {
		printf("[%d]", vet[i]);
	}
}
```

7 - Faça um procedimento recursivo que imprima os elementos da diagonal principal de uma matriz quadrada (se a matriz não for quadrada, imprima essa informação).

```
/*printTabs: imprime uma quantidade de espaços em branco (n * 3)*/
void printTabs (int n) {
	int i;
	for (i = 0; i < n; i++) {
		printf("   ");
	}
}
/*PrintMatrizRec: Imprime os elementos da diagonal principal de uma
matriz quadrada - Usa Recursividade*/
void PrintMatrizRec (int tamLin, int tamCol, int
mat[tamLin][tamCol], int lin) {
	if (tamLin != tamCol) {
		printf("Essa nao e uma matriz quadrada.");
		return;
	}
	else {
		if (lin < 0 || lin >=tamLin) {
			return;
		}
		else {
			printTabs(lin);
			printf("[%d]\n", mat[lin][lin]);
			return PrintMatrizRec(tamLin, tamCol, mat, lin+1);
		}
	}
}
```

8 - Faça um procedimento iterativo que imprima os elementos da diagonal principal de uma matriz quadrada (se a matriz não for quadrada, imprima essa informação).

```
/*PrintMatriz: Imprime os elementos da diagonal principal de uma
matriz quadrada*/
void PrintMatriz(int tamLin, int tamCol, int mat[tamLin][tamCol]){
      int lin;
      if (tamLin != tamCol) {
            printf("Essa nao e uma matriz quadrada.");
      return;
      }
      else {
            for (lin = 0; lin < tamLin; lin++) {
                  printTabs(lin);
                  printf("[%d]\n", mat[lin][lin]);
            }
      }
}
```

9 – Faça uma função recursiva que retorne a soma dos elementos de um vetor.

```
/*SumArrayElementsRec: retorna a soma dos elementos de um vetor -
Usa Recursividade*/
int SumArrayElementsRec (int vet[], int tam, int pos) {
      if (pos < 0 || pos >= tam) {
            return 0;
      }
      else {
            return vet[pos] + SumArrayElementsRec(vet, tam, pos+1);
      }
}
```

10 – Faça uma função iterativa que retorne a soma dos elementos de um vetor.

```
/*SumArrayElements: retorna a soma dos elementos de um vetor*/
int SumArrayElements(int vet[], int tam) {
      int soma=0, i;
      for (i = 0; i < tam; i++) {
            soma += vet[i];
      }
      return soma;
}
```

11 –Faça uma função recursiva que retorne a quantidade de ocorrências de um elemento em um vetor.

```
/*CountElementsArrayRec: retorna a quantidade de ocorrencias de um
elemento em um vetor - Usa Recursividade*/
int CountElementsArrayRec (int vet[], int tam, int pos, int value)
{
      int cont;
      if (pos < 0 || pos >= tam) {
            return 0;
      }
      else {
            cont = ( vet[pos] == value ) ? 1 : 0;
            return cont + CountElementsArrayRec(vet, tam, pos+1,
      value);
      }
}
```

12 –Faça uma função iterativa que retorne a quantidade de ocorrências de um elemento em um vetor.

```
/*CountElementsArray: retorna a quantidade de ocorrencias de um
elemento em um vetor*/
int CountElementsArray (int vet[], int tam, int value) {
      int i, cont = 0;
      for (i = 0; i < tam; i++) {
            cont += ( vet[i] == value ) ? 1 : 0;
      }
      return cont;
}
```

13 – Faça uma função recursiva que retorne a posição da primeira ocorrência de um elemento no vetor.

```
/*IndexOfRec: retorna o index da primeira ocorrencia de um elemento
em um vetor - Recursiva*/
int IndexOfRec(int vet[], int tam, int pos, int value) {
      if ( pos < 0 || pos >= tam ) {
            return -1;
      }
      else {
            return ( vet[pos] == value ) ? pos : IndexOfRec(vet,
      tam, pos+1, value);
      }
```

```
}
```

14 – Faça uma funçao iterativa que retorne a posição da primeira ocorrência de um elemento no vetor.

```
/*IndexOf: retorna o index da primeira ocorrencia de um elemento em
um vetor*/
int IndexOf (int vet[], int tam, int value) {
      int i;
      for (i = 0; i < tam; i++) {
            if ( vet[i] == value ) return i;
      }
      return -1;
}
```

15 – Faça uma função recursiva que calcule o MDC (Máximo Divisor Comum) entre 2 números.

```
/*MDCRec: retorna o MDC de dois numeros - Recursiva*/
int MDCRec (int n1, int n2) {
      int maior, menor, resto;
      maior = (n1 > n2) ? n1 : n2;
      menor = (n1 > n2) ? n2 : n1;
      resto = maior % menor;
      return ( resto == 0 ) ? menor : MDCRec(menor, resto);
}
```

16 – Faça uma função iterativa que calcule o MDC (Máximo Divisor Comum) entre 2 números.

```
/*MDC: retorna o MDC de dois numeros*/
int MDC (int n1, int n2) {
      int maior, menor, resto;
      maior = (n1 > n2) ? n1 : n2;
      menor = (n1 > n2) ? n2 : n1;
      do {
            resto = maior % menor;
            if (resto != 0) {
                  maior  = menor;
                  menor = resto;
            }
      } while (resto != 0);
      return menor;
}
```

17 – Faça uma função recursiva que verifique se os elementos de um vetor formam um palíndromo.

```
/*IsPalindrome: retorna 1 se vetor for palindromo e 0 se não for - Recursiva*/
int IsPalindromeRec (int vet[], int tam, int pos) {
	int middle = (tam > 0) ? ( (int)tam / (int)2 ) : 0;
	if (middle == 0) {
		return 0;
	}
	else if ( pos < 0 || pos >= middle) {
		return;
	}
	else {
		return ( vet[pos] == vet[(tam-1)-pos] ) ? 1 && IsPalindromeRec(vet, tam, pos+1) : 0;
	}
}
```

18 – Pesquise sobre a Torre de Hanoi. Explique o que é e implemente uma solução recursiva
em C.

```
/*HanoiRec: Imprime os passos para realizar a torre de hanoi - recursiva*/
void HanoiRec (char O, char D, char T, int n) {
	if (n > 0) {
		HanoiRec(O, T, D, n-1);
		printf("%c -> %c\n", O, D);
		HanoiRec(T,D,O, n-1);
	}
}
```