

Exercícios

- 1) Faça uma função recursiva para elevar um valor *base* a potência do valor *expoente*.

```
int potenciacao (int base, int expoente) {  
    if (expoente < 1)  
        return 1;  
    return (base * potenciacao(base, expoente-1));  
}
```

- 2) Faça uma função recursiva que calcule o M.D.C. (máximo divisor comum) entre dois valores fornecidos pelo usuário. Por exemplo: m.d.c. de 12 e 20 é 4.

12 , 20	20 - 12 = 8
12 , 8	12 - 8 = 4
4 , 8	8 - 4 = 4
4 , 4	M.D.C.

```
int mdc (int a, int b) {  
    if (a == b) {  
        return a;  
    }  
    if (a > b) {  
        return mdc (a-b, b);  
    } else {  
        return mdc (a, b-a);  
    }  
}
```

- 3) Faça uma função recursiva que calcule o M.D.C. pelo algoritmo de Euclides. Por exemplo, divide-se a por b e obtêm-se o quociente q e o resto r. Se r for zero, b é o m.d.c; se não for, divide-se b por r e assim sucessivamente até encontrar um resto zero. O último divisor é o M.D.C.

```
int mdc (int a, int b) {  
    if (a % b == 0) {  
        return b;  
    }  
    return mdc (b, a % b);  
}
```

- 4) Faça um programa que receba um valor n, e imprima a contagem regressiva a partir deste valor. Por exemplo, se o usuário digitar 5, o programa irá imprimir 5, 4, 3, 2, 1, 0.

```
void contagem (int num) {  
    cout<<num<<"\t";  
  
    if (num != 0) {  
        contagem (num-1);  
    }  
}
```

- 5) Faça uma função recursiva que recebe um vetor preenchido e a quantidade de posições deste vetor, e retorna a soma de todos os elementos do vetor.

Exercícios

```
template <typename tipo>
tipo soma (tipo vet[TAM], int pos) {
    if (pos < 0) {
        return 0;
    }
    return vet[pos] + soma (vet, pos-1);
}
```

- 6) Faça uma função recursiva que realize a multiplicação entre dois valores int, sem utilizar o operador *.

```
int multiplica (int a, int b) {
    if (b < 1) {
        return 0;
    }
    return a + multiplica (a, b-1);
}
```

- 7) Faça uma função recursiva que receba os valores de X e N e realize o cálculo da seguinte progressão geométrica:

$$1 + x + x^2 + x^3 + x^4 + \dots + x^n$$

```
int potenciacao (int base, int expoente) {
    if (expoente < 1)
        return 1;
    return (base * potenciacao(base, expoente-1));
}
```

```
float calculo (int x, int n) {
    if (n < 1) {
        return 1;
    }
    return potenciacao (x, n) + calculo (x, n-1);
}
```

- 8) Faça o teste de mesa do algoritmo recursivo abaixo (procure fazer a mão, e não utilizando o computador), e responda a pergunta:

```

1  #include <iostream>
2  using namespace std;
3
4  int XXX (int n, int m) {
5      if ((n == m) or (m == 0)) {
6          return 1;
7      }
8      return (XXX (n-1, m) + XXX (n-1, m-1));
9  }
10
11 int main() {
12     int n = 5, m = 3;
13     cout << XXX (n, m);
14     return 0;
15 }
```

Exercícios

Qual o valor de $x(5,3)$? = **10**

- 9) Faça o teste de mesa do programa abaixo (procure fazer a mão, e não utilizando o computador), e informe o resultado final. = **20**

```

1  #include <iostream>
2  using namespace std;
3
4  int qualquer (int n) {
5      if (n<=2) {
6          return n;
7      }
8      return (qualquer(n-1) + qualquer(n-2) + qualquer(n-3));
9  }
10
11 int main () {
12     int n = 6;
13     cout<<qualquer(n);
14     return 0;
15 }
```

- 10) Escreva um programa que leia quatro valores inteiros positivos n_A , n_B , t_A e t_B , representando respectivamente as populações atuais de dois países A e B e as taxas de crescimento anual dessas populações, e determine se o país menos populoso poderá ultrapassar a população do outro país, supondo que as taxas de crescimento dessas populações não variam. Em caso afirmativo, o programa deverá determinar também o número de anos necessários para que isto aconteça. Utilize funções recursivas para resolver o problema.

```

#include <iostream>
using namespace std;

int crescimento (float nA, float nB, float tA, float tB, int anos) {
    if (nA < nB) {
        return anos;
    }
    return crescimento (nA * (1 + tA/100), nB * (1 + tB/100), tA, tB, anos
+ 1);
}

int main() {

    float nA, nB, tA, tB;

    cout<<"Populacao do Pais A .....: ";
    cin>>nA;
    cout<<"\nTaxa de crescimento do Pais A ..: ";
    cin>>tA;

    cout<<"\nPopulacao do Pais B .....: ";
    cin>>nB;

```

Exercícios

```
cout<<"\nTaxa de crescimento do Pais B ...: ";
cin>>tB;

if ((nA > nB) && (tA > tB)) {
    cout<<"\nO pais B nunca ultrapassara o pais A";
} else if ((nA < nB) && (tA < tB)) {
    cout<<"\nO pais A nunca ultrapassara o pais B";
} else if ((nA == nB) && (tA == tB)) {
    cout<<"\nOs dois possuem a mesma taxa e mesma populacao";
} else if ((nA > nB) && (tA < tB)){
    cout<<"\nLevara  %d anos para o pais B ultrapassar o pais
A",crescimento(nA,nB,tA,tB,0);
} else if ((nA < nB) && (tA > tB)){
    cout<<"\nLevara  %d anos para o pais A ultrapassar o pais
B",crescimento(nB,nA,tB,tA,0);
}
return 0;
}
```

- 11) Escreva uma função recursiva, `int SomaSerie (int i, int j, int k)`, que imprime na tela a soma de valores do intervalo `[i,j]`, com incremento `k`.

```
float calculo (int i, int j, int k) {
    if (i > j) {
        return 0;
    }
    return i + calculo (i + k, j, k);
}
```

- 12) Faça um programa recursivo que contenha uma função para calcular e retornar o resultado da seguinte série:

$$\frac{1}{N} + \frac{2}{N-1} + \frac{3}{N-2} + \frac{4}{N-3} + \dots$$

N é um valor inteiro maior ou igual a 1, digitado pelo usuário. A série deve ser calculada até que o denominador seja igual a 1. O valor de N deverá ser fornecido pelo usuário.

```
float calculo (int num, int cont) {
    if (num < 1) {
        return 0;
    }

    return num/float(cont) + calculo (num-1, cont+1);
}
```

- 13) A **pesquisa** ou **busca binária** (em inglês *binary search algorithm* ou *binary chop*) é um algoritmo de busca em vetores que requer acesso aleatório aos elementos do mesmo. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca (divisão e conquista) comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o

Exercícios

elemento do meio vier depois da chave, a busca continua na metade anterior do vetor (Fonte: Wikipédia).

Abaixo, segue um exemplo de pesquisa binária (ou busca binária) sem usar recursividade:

```
1  #include <iostream>
2  using namespace std;
3
4  #define TAM 10
5
6  int PesquisaBinaria ( int vet[], int elemento , int tamanho_vetor) {
7      int limite_inferior = 0, limite_superior = tamanho_vetor-1, meio;
8      while (limite_inferior <= limite_superior) {
9          meio = (limite_inferior + limite_superior)/2;
10         if (elemento == vet[meio]) {
11             return meio;
12         } else if (elemento < vet[meio]) {
13             limite_superior = meio-1;
14         } else {
15             limite_inferior = meio+1;
16         }
17     }
18     return (-1);
19 }
20
21 int main () {
22     int vet[TAM]={1,2,3,4,5,6,7,8,9,10}, guarda, elemento;
23     cout<<"Entre com o valor que deseja procurar no vetor ...: ";
24     cin>>elemento;
25     guarda = PesquisaBinaria(vet,elemento,TAM);
26     if (guarda == -1) {
27         cout<<"\n\nO elemento "<<elemento <<" nao foi encontrado";
28     } else {
29         cout<<"\n\nO elemento "<<elemento<<" foi encontrado na posicao "<<guarda;
30     }
31     return 0;
32 }
```

Faça um programa de busca binária recursivo. Crie um vetor de 15 posições e preencha-o utilizando rand, e cuidando para não haver valores repetidos. A seguir ordene-o (tanto o preenchimento quanto a ordenação não precisam ser recursivos) e solicite ao usuário que entre com o valor que deseja procurar no vetor. Esse valor será levado a uma função de busca binária recursiva, que irá retornar verdadeiro se o elemento existir no vetor e falso se não existir. Exiba esta mensagem na tela.

```
bool PesquisaBinaria (int vet[], int elemento, int posicao_inicial, int
posicao_final) {

    if (posicao_inicial > posicao_final) {
        return false;
    }

    int meio = (posicao_inicial + posicao_final)/2;
```

Exercícios

```

    if (elemento < vet[meio]) {
        return PesquisaBinaria (vet, elemento, posicao_inicial, meio - 1);
    }

    if (elemento > vet[meio]) {
        return PesquisaBinaria (vet, elemento, meio+1, posicao_final);
    }

    return true;
}

```

- 14) Faça uma função recursiva, em linguagem C, que calcule o valor da série S descrita a seguir para um valor $n > 0$ a ser fornecido como parâmetro para a mesma.

$$S = 2 + \frac{5}{2} + \frac{10}{3} + \frac{17}{4} + \dots + \frac{1 + n^2}{n}$$

```

int potenciacao (int base, int expoente) {
    if (expoente < 1) {
        return 1;
    }
    return base * potenciacao(base, expoente-1);
}

float calculo (int n) {
    if (n < 1) {
        return 0;
    }
    return (1 + potenciacao(n, 2)) / float(n) + calculo (n - 1);
}

```

- 15) Pode-se calcular o resto da divisão, MOD, de x por y, dois números inteiros, usando-se a seguinte definição:

$$MOD(x, y) = \begin{cases} MOD(|x| - |y|), & \text{se } |x| > |y| \\ |x|, & \text{se } |x| < |y| \\ 0 & \text{se } |x| = |y| \end{cases}$$

Então, pede-se que seja criada uma função recursiva para descrever tal definição. A função deve retornar -1 caso não seja possível realizar o cálculo. Além disso, crie um algoritmo que leia os dois valores inteiros e utilize a função criada para calcular o resto da divisão de x por y, e imprima o valor computado.

```

int MOD (int x, int y) {
    if ((x < 0) || (y < 0)) {
        return -1;
    }
    if (x > y) {
        return MOD (x-y, y);
    } else if (x < y) {

```

Exercícios

```
    return x;  
}  
return 0;  
}
```

- 16) Pode-se calcular o quociente da divisão, DIV, de x por y, dois números inteiros, usando-se a seguinte definição:

$$DIV(x, y) = \begin{cases} 1 + DIV(|x| - |y|, |y|), & \text{se } |x| > |y| \\ 0, & \text{se } |x| < |y| \\ 1, & \text{se } |x| = |y| \end{cases}$$

Então, pede-se que seja criada uma função recursiva para descrever tal definição. A função deve retornar -1 caso não seja possível realizar o cálculo. Além disso, crie um algoritmo que leia os dois valores inteiros e utilize a função criada para calcular o quociente de x por y, e imprima o valor computado.

```
int DIV (int x, int y) {  
    if ((x < 0) || (y < 0)) {  
        return -1;  
    }  
    if (x > y) {  
        return (1 + DIV (x-y, y));  
    } else if (x < y) {  
        return 0;  
    }  
    return 1;  
}
```

- 17) Um problema típico em ciência da computação consiste em converter um número da sua forma decimal para a forma binária. Por exemplo, o número 12 tem a sua representação binária igual a 1100. A forma mais simples de fazer isso é dividir o número sucessivamente por 2, onde o resto da i-ésima divisão vai ser o dígito i do número binário (da direita para a esquerda).

Por exemplo:

12 / 2 = 6, resto 0 (1º dígito da direita para esquerda)

6 / 2 = 3, resto 0 (2º dígito da direita para esquerda)

3 / 2 = 1 resto 1 (3º dígito da direita para esquerda)

1 / 2 = 0 resto 1 (4º dígito da direita para esquerda).

Resultado: 12 = 1100.

Exercícios

Faça uma função recursiva que dado um número decimal imprima a sua representação binária corretamente.

```
long long int Dec2Bin (int n, int cont, long long int bin) {
    if (n < 1) {
        return bin ;
    }
    return Dec2Bin (n/2, cont*10, bin + cont * (n % 2));
}

int Bin2Dec(long long int bin, int pot, int dec) {
    if (bin < 10) {
        return (dec + pow(2,pot));
    }
    return Bin2Dec(bin/10, pot + 1, bin%10 == 1?dec + (pow(2,pot)):dec);
}
```