

Material de apoio para disciplina GCC224 - Introdução aos Algoritmos

Período: ERE 2020

Neste material você encontrará dicas para pensar antes de escrever uma solução, códigos discutidos passo-a-passo.

Não pense que as sugestões de códigos a seguir são as melhores soluções. Sempre se pergunte como você faria. Se quiser, compartilhe conosco sua solução.

Pensando em casos de testes

Perguntas que devem ser feitas quando se está pensando em casos de teste

Resolvendo problema numérico

- 1) O tipo dos dados estão corretos de acordo com o enunciado?
- 2) A ordem de entrada dos dados está correta?
- 3) Poderá ocorrer entrada de dados iguais?
- 4) Poderá ocorrer entrada de valor igual a zero?
- 5) Poderá ocorrer entrada de dados negativos?
- 6) Poderá ocorrer divisão por zero?
- 7) Há divisão de dois inteiros resultando em um novo inteiro?
- 8) As casas decimais precisam ser tratadas?
- 9) Há arredondamento?
- 10) Há tratamento de intervalos de dados (Ex: $1 \leq x < 10$) onde preciso testar os limites?
Todos os intervalos foram contemplados?
- 11) Há tratamento diferente para números pares e ímpares?
- 12)

Resolvendo problema que envolve escrita de string

- 1) A saída deve ser em letra maiúscula ou minúscula?

Cuidados com números int, float e divisões

Código	Saída
<pre>int a,b,c; float p; p=(a+b+c)/2;</pre>	Entrada: 2 3 4 Saída de p: 4 Mesmo p sendo float, a divisão de variáveis inteiras, gera um valor inteiro, truncando a

	parte decimal.
<pre>int a,b,c; float p; p=float(a+b+c)/2;</pre>	Entrada: 2 3 4 Saída de p: 4.5 É feito um casting para a saída ser de fato um float
<pre>int a,b,c; float p; p=(a+b+c)/2.0;</pre>	Entrada: 2 3 4 Saída de p: 4.5 É feito um casting para a saída ser de fato um float
<pre>float p=2/3;</pre>	Saída será 0.
<pre>float p=2/3.0;</pre>	Saída será 0.666667
<pre>float soma = 0.0; int num = 0; cout<<soma/num<<endl;</pre>	Terá como saída -nan Se for cout<<0/0<<endl; divisão de 2 inteiros, resulta em Floating point exception (core dumped)

Dredd disse que meu programa tem trechos perigosos

Muitas vezes sua lógica está correta, mas o Dredd te retorna: *Existem trechos perigosos no programa*. Você se pergunta: Onde? Como assim?

Veja algumas situações em que isso pode acontecer.

- **declarar variável e não utilizá-la**

Exemplo:

```
int x,i;
cin >>x;
for (int i=0;i<x;i++)
    cout<<i<<endl;
```

No código acima, a variável da primeira linha nunca é utilizada. Para complicar, ainda tem outra variável i declarada no for. Ou seja, temos 2 variáveis i, mas só uma está sendo usada. Para corrigir, deveria tirar o primeiro i ou tirar int do for. Não esqueça que variáveis ocupam memória, memória custa caro dependendo da situação, variáveis podem guardar lixo de memória.

- **função que não alcança nenhum return**

Exemplo:

```
int funcao(int y) {
    if(y==0)
        return y;
```

```

else if(y>1)
    return funcao(y-10)+funcao(y-20);

}

```

Uma função tem que retornar algum valor em qualquer situação. No exemplo acima, se porventura *y* não passar em nenhuma das condições, a função terá um comportamento inesperado. Esse trecho compila e executa, mas o compilador emitirá um warning (*control reaches end of non-void function [-Wreturn-type]*). Uma forma de resolver, é deixar o último return fora da condição, isso garantirá que qualquer outro valor diferente de 0, entrará em algum return.

Exemplos de problemas com estrutura de repetição

1. Cálculo do MDC.

Exemplo com **while** usando o código do slide (Estrutura de repetição, slide 18).

Teste o código com vários valores. SEMPRE TESTE!!

```

#include <iostream>
using namespace std;
int main(){
    int A, B, MDC, MENOR, MAIOR;
    cin >> A >> B;

    if (A >= B) {
        MAIOR = A;
        MENOR = B;
    } else {
        MAIOR = B;
        MENOR = A;
    }

    while ((MAIOR % MENOR) != 0) {
        MDC = MAIOR % MENOR;
        MAIOR = MENOR;
        MENOR = MDC;
    }

    cout << MDC << endl;
    return 0;
}

```

Executando o código com os dados da tabela a seguir:

Entrada: A e B	Saída do código	Saída esperada

25 80	5	5
89 9	1	1
300 200	100	100
20 5	21928	5
3 6	22027	3

A explicação a seguir parte da ideia que você entendeu o cálculo do MDC como explicado na videoaula.

Observações:

1. O bloco if/else sempre será executado e o seu resultado é colocar os valores corretos nas variáveis MAIOR e MENOR.
2. O critério de parada da repetição é *enquanto maior dividido por menor tiver resto diferente de 0*.
3. Deu certo nas 3 primeiras linhas da tabela. Porque não deu nas 2 últimas? O que elas tem de diferente?

Na primeira tentativa de rodar o while não passa no critério de parada e vai direto para a linha do cout.

Como? Ora, 20 (MAIOR) % 5 (MENOR) não tem resto diferente de 0. 6 (MAIOR) % 3 (MENOR) não tem resto diferente de 0. Sacas???

4. Quando não entra no while, aparece um número esquisito na saída.

Porque a variável MDC não foi inicializada. Aí, quando chega no cout, o computador escolhe o primeiro bagulho (lixo de memória) que encontrar no caminho e mostra na tela.

Há várias formas de resolver esse problema. Uma forma simples é tratar o caso de quando os valores que chegam logo de primeira são uma divisão exata de um pelo outro. Observe que nesse caso, o MDC é o menor valor. Podemos inicializar a variável MDC com o valor da variável MENOR. Isso afeta o resultado de quando não for divisão exata, que é o caso de 300%200?

De forma nenhuma.

A seguir o código com uma pequena alteração.

```
#include <iostream>
using namespace std;
int main(){
    int A, B, MDC, MENOR, MAIOR;
    cin >> A >> B;

    if (A >= B) {
        MAIOR = A;
        MENOR = B;
    } else {
```

```

        MAIOR = B;
        MENOR = A;

    }
    MDC=MENOR; //linha inserida
    while ((MAIOR % MENOR) != 0) {

        MDC = MAIOR % MENOR;
        MAIOR = MENOR;
        MENOR = MDC;
    }

    cout << MDC << endl;
    return 0;
}

```

2. Exemplo de while com acumuladores e contadores. (Estrutura de repetição, slide 37).

Problema: construir um programa em C++ que leia um número inteiro positivo não nulo N, que indica a quantidade de elementos de uma sequência de números reais que será fornecida em seguida. Seu programa deverá calcular e exibir a média aritmética dos números reais da sequência fornecida.

Note que, neste exemplo, precisaremos de duas variáveis auxiliares: uma variável contadora (para contabilizar quantos números foram utilizados durante o processo de leitura) e uma variável acumuladora (para somar todos os números reais).

Observações a partir do código:

Se num for 5, significa que teremos que digitar na sequência cinco valores inteiros.

A variável soma é o acumulador aqui. Qual o papel dela? Acumular a soma de todos os valores lidos na variável valor.

A variável cont é contadora. Por que? Sua missão é controlar a quantidade de valores digitados para a variável valor. A repetição para quando o valor de cont não passar mais na condição cont < num.

No final queremos mostrar a média dos valores lidos. Como calculamos a média?

Uhm, soma todos os elementos e dividimos pela quantidade. Então, quais variáveis guardam essas informações para nós? Muito bem, são as variáveis soma e num.

```

#include <iostream>
using namespace std;

int main() {
    int num;
    int cont = 0;
    float valor;
    float soma = 0;
    cin >> num; // lendo a quantidade de numeros
    while (cont < num) {
        cin >> valor; // lendo um valor
    }
}

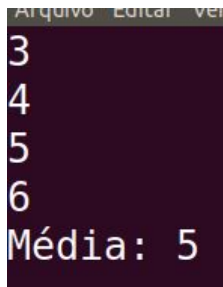
```

```

    cont++;          // contador
    soma += valor;   // acumulador
}
cout << "Média: " << float(soma)/num << endl;
return 0;
}

```

Execução passo a passo

cont	soma	num	cont < num	valor	float (soma/num)	Tela do terminal
0	0	3	V	4		
1	4		V	5		
2	9		V	6		
3	15		F		15/3 = 5	

Pergunta: por que a variável soma começa de 0? Para que no final da repetição essa variável tenha a soma de todos os valores digitados, precisamos a cada rodada fazer com que ela receba o valor atual + soma anterior. No exemplo, quando cont=0, soma recebe 0+4. Quando cont=1, soma recebe 4 (que é a soma anterior) + o valor atual, ou seja 4+5, guardando 9 em soma. Na próxima rodada, cont será 2, soma receberá o que ela já tem que é 9 + valor atual, ou seja, 9+6, guardando 15 em soma.

Pergunta: por que a variável cont começou de 0? Essa variável é utilizada no controle do laço, serve para contar a quantidade de valores lidos. Na repetição, a variável utilizada no controle de parada precisa ter algum valor antes de alcançar a linha do while (neste caso é 0, mas poderia ser outro valor).

Pergunta: por que a variável cont tem ++? ++ é equivalente a cont = cont +1. Essa instrução indica que a variável cont crescerá de 1 em 1. Veja que essa mesma variável é utilizada no controle de parada do while. Essa instrução fará com que a variável cont se aproxime de num. Se você não colocar essa instrução, cont sempre será 0 e entramos em loop infinito.

Uma repetição sempre tem um início, um critério de parada e uma instrução que faz com que você se aproxime do critério de parada.

E se fosse a multiplicação de todos os valores lidos, que mudanças **você** faria no código?

3. Exemplo: Imprimir os números naturais de 0 até n.

Código iterativo:

```
#include <iostream>
using namespace std;

int main() {
    int a, cont=0;
    cin>>a;
    while (cont<=a) {
        cout<<cont<<" ";
        cont++;
    }
    cout<<endl;

    return 0;
}
```

Exemplos de problemas com Modularização

1. Problema: **Imprimir os números naturais de 0 até n.**

Lembre que entrada de dados geralmente fica no módulo principal.

```
#include <iostream>
using namespace std;

void imprime(int a) {
    int cont=0;
    while (cont<=a) {
        cout<<cont<<" ";
        cont++;
    }
    cout<<endl;
}

int main() {
    int a;
    cin>>a;
    imprime(a);
    return 0;
}
```

Exemplos de problemas com Recursão

1. Exemplo com recursividade: Número áureo

O *número áureo*, frequentemente denotado pela letra grega ϕ (phi) é um número real irracional que ocorre espontaneamente na natureza e é frequentemente usado nas artes por estar relacionado à nossa percepção de beleza.

O *número áureo* pode ser calculado pela recorrência $\phi = 1 + 1/\phi$.

Por ser uma recorrência infinita, ela precisa ser limitada para ser usada na recursividade da Computação. Podemos definir o valor aproximado de ϕ em função do número de termos usados no cálculo, assim:

Faça um programa que tem uma função que calcula uma aproximação do *número áureo*, usando recursão.

O *número áureo*, deve ser do tipo *ponto flutuante de precisão dupla* (double) para possibilitar a precisão necessária nos cálculos. As operações de leitura e escrita devem ser realizadas na função principal.

Entradas:

1. O número de termos para o cálculo da aproximação do *número áureo*.

Saídas:

1. O valor aproximado do *número áureo*.

Exemplo de Entrada:

3

Exemplo de Saída:

1.5

1) Entendendo o problema

Dado um valor de n queremos calcular o valor da função $\phi(n)$ recursivamente.

Para implementar um algoritmo recursivo, precisamos entender o problema para identificar qual é o case base (subprograma não chama a si mesmo), o caso geral e como será a chamada recursiva.

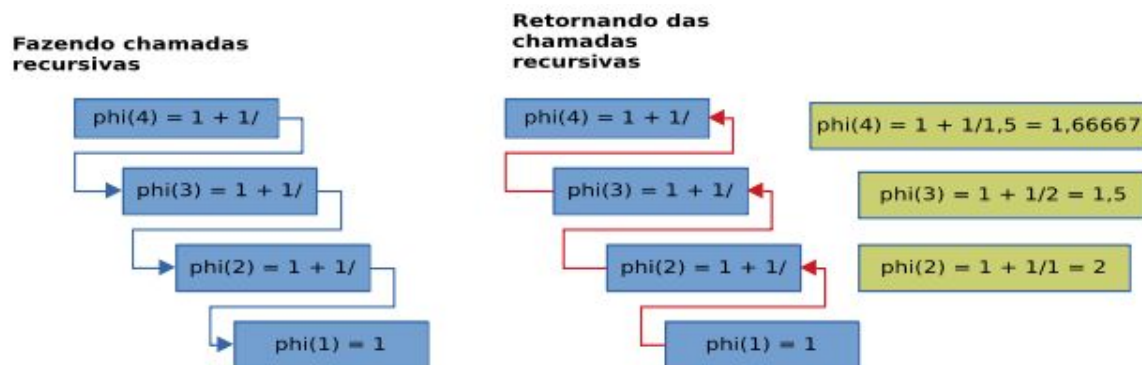
Nesse problema, o caso base é quando $n=1$, pois sabemos o valor da função $\phi(1)=1$.

O caso geral é $\phi(n) = 1 + 1/\phi(n-1)$.

A chamada recursiva é $\phi(n-1)$, pois está chamando a mesma função $\phi()$ passando o valor de $n-1$.

Vamos considerar um exemplo onde $n=4$ e as chamadas realizadas que são ilustradas na Figura 1.

Figura 1 – Chamadas recursivas para $\phi(4)$



Começamos chamando a função $\phi(n)$, como $n=4$ fica:

$$\phi(4) = 1 + 1/\phi(4-1) = 1 + 1/\phi(3)$$

Para continuarmos resolvendo $\phi(4)$ precisamos descobrir o valor de $\phi(3)$:

$$\phi(3) = 1 + 1/\phi(3-1) = 1 + 1/\phi(2)$$

Novamente, temos que calcular o valor de ϕ de um número ($\phi(2)$) antes de encontrar o resultado que precisamos.

$$\phi(2) = 1 + 1/\phi(2-1) = 1 + 1/\phi(1)$$

Ainda não chegamos no resultado, temos que calcular $\phi(1)$.

Na definição da função ϕ , temos que $\phi(1)=1$. Agora chegamos no caso base, ou seja, no caso onde sabemos o valor da função $\phi(1)$. Precisamos voltar na chamada da função $\phi(2)$ que precisava do valor de $\phi(1)$:

$$\phi(2) = 1 + 1/\phi(1) = 1 + 1/1 = 2$$

Obtivemos o valor de $\phi(2)$, precisamos voltar em $\phi(3)$ e substituir esse valor:

$$\phi(3) = 1 + 1/\phi(2) = 1 + 1/2 = 1,5$$

Fazemos o mesmo para $\phi(4)$:

$$\text{phi}(4) = 1 + 1/\text{phi}(3) = 1 + 1,5 = 2,5$$

Assim chegamos ao resultado.

2) Pensando no algoritmo.

Qual será a entrada? Será o valor de n informado pelo usuário. Então temos que fazer a leitura do valor e armazenar em uma variável.

Declara variável **n**

Lê valor e armazena em **n**

Em C++:

```
int n;
```

```
cin >> n;
```

Qual será a saída? Será o resultado de phi(n). A variável que contém o resultado deverá impressa na tela.

Declara variável **phi**

Imprime valor de **phi** na tela.

Em C++:

```
float phi;
```

```
cout << phi << endl;
```

A variável phi recebe o resultado do cálculo da função.

```
phi = calcula_phi(n);
```

Qual será a função recursiva?

Será a função calcula_phi(n).

Assim, no programa principal teremos que chamar calcula_phi(n) para calcular o valor

Analisando phi(n), percebemos que podemos escrever:

se (n==1) então

```
phi = 1
```

senão

$\text{phi} = 1 + 1/(\text{chama função passando } n-1)$

retorna phi

Em C++ fica:

```
if (n==1) {  
    phi=1;  
} else{  
    phi = 1 + 1/calcula_phi(n-1);  
}  
return phi;
```

3) Fazendo a declaração e passagem de parâmetros:

```
float calcula_phi(int n){  
    float phi;  
    if (n==1) {  
        phi=1;  
    } else{  
        phi = 1 + 1/calcula_phi(n-1);  
    }  
    return phi;  
}
```

Chamada da função calcula_phi() na função principal.

```
int main(){  
    int n;  
    cin >> n;  
    cout << calcula_phi(n) << endl;  
    return 0;  
}
```

4) Quais são os casos de teste que devo testar?

Entrada: N	Saída do código	Saída esperada
------------	-----------------	----------------

1		1
2		2
3		1.5
4		1.66667
5		1.6
6		1.625
7		1.61538

O enunciado não pede, mas poderia ser tratada a situação onde o valor de $n \leq 0$.

```

if (n <= 0)
    cout << "valor inválido de n" << endl;
else {
    //faz calculo de phi(n)
}

```

2. Exemplo com recursividade: Quantos dígitos um número possui

Crie uma função recursiva que calcula quantos dígitos um número possui. Para saber quantos dígitos um número possui basta ir dividindo por 10 até o quociente chegar em zero.

Exemplo: $n=1555$

Número	Contador de dígitos
$1555/10 = 155$	1

$155/10 = 15$	$1+1$
$15/10=1$	$1+ 1 + 1$
$1/10 = 0$	$1 + 1 + 1 + 1 = 4$

Este número tem 4 dígitos.

Resolvendo iterativamente com estrutura de repetição while.

Entrada: n

Saída: número de dígitos do número n

Condição de parada $n/10 == 0$

```
int main(){
    int n, cont=1;
    cin >> n;
    while (n/10 !=0){
        cont++;
        n=n/10;
    }
    cout << cont << endl;
return 0;
}
```

Usando modularização: o while vai do main para uma função digito()

```
int digito(int n){
    int cont=1;
    while (n/10 !=0){
        cont++;
        n=n/10;
    }
    return cont;
}
int main()
{
    int n, cont=1;
    cin >> n;
    cout << digito(n);
    return 0;
}
```

Usando recursividade.

O case base é quando ao fazer $n/10$ o quociente for 0, pois estamos com apenas 1 algarismo.

Se o $n/10 \neq 0$ significa que temos mais de 1 algarismo e ainda precisamos contar quantos tem.

Cada vez que dividimos por 10 temos que contar 1 ocorrência de 1 algarismo e chamar a função recursiva para calcular o número de dígitos para o número resultante de $n/10$.

```
int calcula(int n){
    int resultado;
    if (n/10 == 0) return 1;
    else {
        resultado = 1+calcula(n/10);
        return resultado;
    }
}

int main()
{
    int n, cont=1;
    cin >> n;
    cout << calcula(n);
    return 0;
}
```

Vamos fazer o teste de mesa para $n=1000$ ao chamar `calcula(1000)`. Dentro da função `calcula`, temos as variáveis `n`, `resultado`.

`calcula(1000)`

calcula(1000)	n=1000 n/10 == 0 (F) resultado = 1+calcula(100);	resultado = 1 + 3 = 4 return resultado
calcula(100)	n=100 n/10 == 0 (F) resultado = 1+calcula(10);	resultado = 1 + 2 = 3 return resultado
calcula(10)	n=10 n/10 == 0 (F) resultado = 1+calcula(1);	resultado = 1 + 1 = 2 return resultado
calcula(1)	n=1 n/10 == 0 (V) return 1	

32. Exemplo com recursividade: Calcular logaritmo na base 2

Faça um programa que calcula o logaritmo na base 2 de uma potência de 2. Para tanto, divisões sucessivas deverão ser utilizadas. O programa deve conter um subprograma recursivo que recebe um número natural e retorna um número natural para fazer o cálculo do logaritmo.

Entrada: um número inteiro que é potência de 2.

Saída: logaritmo do número lido.

Exemplo de entrada: 256

Exemplo de saída: 8.

O logaritmo de 256 na base 2 é 8 porque 2 elevado a 8 é 256.

Fazendo sucessivas divisões temos:

256	2 >> lê-se 256 / 2 é 128
128	2
64	2
32	2
16	2
8	2
4	2
2	2
1	0

Observe que o caso base será quando o valor a ser dividido for 1.

As outras divisões serão a chamada recursiva: $\log(n/2)$

```
#include <iostream>
using namespace std;
int log (int n) {
```

```

    if (n==1)
        return 0;
    // return 1 + log (n/2);
    return log (n/2);
}
int main() {
    int n;
    cin>>n;
    cout<<log(n)<<endl;

    return 0;
}

```

Executando. Suponha a=8. Log 8 = 3

<p>1) log (8) n==1 -> F. Cai no último return. log(4)</p>	-> chamada recursiva, empilha	Resolve. Recebe 0. Envia para resultado no cout.
<p>2)log(4). n==0 -> F. Cai no último return. log(2)</p>	-> chamada recursiva, empilha	Resolve. Recebe 0. Leva 0 para a chamada (1)
<p>3)log(2). n==0 -> F. Cai no último return. log(1)</p>	-> chamada recursiva, empilha	Resolve. Recebe 0. Leva 0 para a chamada (2)
<p>4) log(1). n==0 -> V. Resolve. Retorna 0.</p>	-> resolveu. Volta na pilha levando o valor 0 (chamada 3)	

Podemos ver um erro no resultado final. Quando retorna na chamada 3, onde temos o log de 2, ali o resultado deveria ser 1. Quando retorna na chamada 2, onde temos o $\log(4)$, o resultado deveria ser 2. Quando volta na primeira chamada, onde temos $\log(8)$, deveríamos ter 3 como resposta. **Como podemos corrigir?**

Vamos pensar no $\log(2)$. Se já temos 0 de $\log(1)$, podemos somar +1, que resultará em 1. Quando alcançar $\log(4)$, teremos 1 da chamada abaixo na pilha + 1 chamada atual.

Tire a linha que está em comentário no código e comente a próxima, e dará certo.