

DIATINF – Diretoria Acadêmica de Gestão e Tecnologia da Informação

TADS – Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL

Rio Grande do Norte

Campus

Natal-central

Algoritmos

Exercícios Linguagem C++

Professor Jorgiano Marcio Bruno Vidal

Aluno Fernando Yuri Vital de Aquino

1. Sobre o programa, em C++, a seguir:

**#include** <iostream>

**int** soma(**int** a, **int** b) {

**int** r = a + b;

**return** r;

}

**int** main() {

**int** a, b;

std::cin >> a >> b;

**int** s = soma(a, b); std::cout << s << std::endl; **return** 0;

}

* 1. Implemente e compile seu programa.
  2. Execute o programa com os valores a seguir e escreva o resultado.

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **soma** |
| 10 | 20 |  |
| −10 | 10 |  |
| −2 | −3 |  |
| 1000000000 | 1000000000 |  |
| −1000000001 | 1000000005 |  |
| 2000000000 | 1000000000 |  |
| −1212121212 | −2987654321 |  |

* 1. Explique o motivo de cada um dos resultados obtidos.
  2. Pesquise e identifique correções para os resultados incorretos.

1. Escreva um programa que leia dois lados *l*1 e *l*2 que forma um retângulo, como mostrado a seguir, e mostre o perímetro do retângulo.

*l*1

*l*2

Considere a restriçao 1 ≤ *li* ≤ 109, ou seja, tanto *l*1 quanto *l*2 possuem valores entre 1 e 109 (1000000000 - Um bilhão).

* 1. Determine e explique qual o tipo de *l*1 e de *l*2.
  2. Escreva uma função que receba Os 2 (dois) parâmetros e retorne o perímetro. Determine o tipo de dado de retorno da função.
  3. Escreva um conjunto com, pelo menos, 5 casos de teste e monte uma tabela.

1. Escreva um programa que leia um número real, com 5 (cinco) casas decimais, que represente o diâmetro de um círculo e mostre a área *a* do mesmo, de acordo com a fórmula da área a seguir:

*a* = *πr*2

onde *r* é o raio do círculo. Seu programa deve mostrar o resultado com 5 casas decimais.

Em C++ há dois tipos básicos para trabalhar com números reais: float e double. Implemente duas versões do seu programa, uma versão com o tipo float e outra com o tipo double.

Preencha a tabela a seguir com a execução de cada um dos programas para os valores de entrada a seguir:

|  |  |  |
| --- | --- | --- |
| **diâmetro** | float | double |
| 1*.*00000 |  |  |
| 10*.*00000 |  |  |
| 50000*.*00000 |  |  |
| 100000*.*00000 |  |  |
| 200000*.*00000 |  |  |
| 1*.*56780 |  |  |
| 123*.*45678 |  |  |
| 12345678*.*87654 |  |  |
| 2000000000*.*88888 |  |  |

**PS1:** Considere o valor de *π* com 5 casas decimais: 3*.*14159.

**PS2:** Lembre-se que o raio é o diâmetro dividido por 2.

1. Escreva um algoritmo que calcule a quantidade de postes a serem colocados em uma rua. O algoritmo tem como entrada a distância do início ao fim da rua, um número inteiro *m*, em metros, e a distância entre 2 (dois) postes *p*, também em metros. Há sempre um poste no ponto 0 da rua e outro no ponto *m*, que é o final da rua. Seu programa deve mostrar a quantidade de postes a serem colocados e a distância entre os dois últimos postes. Lembre-se que há sempre um poste no início da rua e outro no final. A distância entre cada par de postes é o valor, em metros, lido pelo programa, com exceção da distância entre os dois últimos postes da rua.

*Exemplo:* Se a rua tiver 1000 metros e a distância entre os postes for de 250 metros então temos 5 postes, com distância de 250 metros entre eles. Já se a rua tiver os mesmos 1000 metros e a distância entre os postes for de 300 metros, a rua terá os mesmos 5 postes, mas a distência entre os dois últimos postes será de 100 metros.

1. Escreva um programa leia um número natural positivo *n* (1 ≤ *n* ≤ 109) e mostre a quantidade de divisores de *n*.

* Determine o tipo da variável que deve armaenar *n*.
* Escreva uma função que receba *n* e retorne a quantidade de divisores.
* Faça testes e verifique se seu programa funciona corretamente.

A seguir um algoritmo para calcular a quantidade de divisores de *n* ∈ N+.

**Algorithm 1** Calcula a quantidade de divisores de *n*.

**Entrada:** *n* ∈ N+ **and** 1 ≤ *n* ≤ 109

**Saída:** Quantidade de divisores de *n divs* → 0

**para** *i* → 1 **até** *n* − 1 **faça**

**se** *n* mod *i* = 0 **então**

*divs* → *divs* + 1

# fim se

**fim para retorne** *divs*

1. No conjunto de números naturais positivos N+, um número *n* é considerado primo quando possui exatamente dois divisores no cojunto, o número 1 e o próprio *n*. O número 1 não é considerado primo, pois possui apenas 1 divisor, que é ele mesmo. O número 2 é considerado primo, pois possui exatamente 2 divisores no conjunto N+. O 2 é também o único número par que é primo. Exemplos de números primos são 3, 5, 7, 11, 13, etc. Uma das formas mais simples de determinar se um número é primo é contar quantos divisores o número tem, entre os possíveis divisores, e verificar se a quantidade é igual a 2. Escreva uma função, em C++, que rebeba um número *n* ∈ N+

(1 ≤ *n* ≤ 109) e retorne um valor lógico (*bool*) informando se *n* é primo.

Segue um algoritmo para determina se um número é primo:

**Algorithm 2** Determina se *n* é primo.

**Entrada:** *n* ∈ N+ ∧ 1 ≤ *n* ≤ 109

**Saída: verdadeiro** se *n* é primo, **falso** caso contrário.

*divs* → 0

**para** *i* → 1 **até** *n* − 1 **faça se** *n* mod *i* = 0 **então**

*divs* → *divs* + 1

# fim se

**fim para retorne** *divs* = 2

**PS:** Existem algoritmos mais eficientes de se determinar se um número é primo. O objetivo deste exercícios é ler um algoritmo e passar para a linguagem de programação C++.

1. Sejam *a* e *b* dois números naturais positivos, eles são considerados **primos entre sí** se não possuírem divisores em comum além do número 1 (um). A seguir um algoritmo para determinar se dois números são primos entre sí:

**Algorithm 3** Determina *a* e *b* são primos entre sí.

**Entrada:** *a* ∈ N+ ∧ 1 ≤ *a* ≤ 109

**Entrada:** *b* ∈ N+ ∧ 1 ≤ *b* ≤ 109

**Saída: verdadeiro** se *a* e *b* são primos entre sí. **falso** caso contrário.

*primos* → **verdadeiro**

*n* → *menor*(*a, b*)

**para** *i* → 2 **até** *n* **faça**

**se** *a* mod *i* = 0 ∧ *b* mod *i* = 0 **então**

*primos* = **falso**

# fim se fim para

**retorne** *primos*

Escreva, em C++, um programa para verificar se 2 números são primos entre sí. Escreva testes para seu programa.

1. Escreva um algoritmo para determinar o MDC (máximo divisor comum) entre 2 números naturais positivos *a* e *b*. Implemente e teste seu algoritmo em C++.

A seguir 2 sugestões de ideias de algoritmos:

* 1. Começar um laço do menor número e decrementar até encontrar um divisor comum. Ao encontrar, para o laço e retorna o valor encontrado. No pior caso o valor retornado será o número 1, que é divisor de todos os números naturais positivos.
  2. Definir o máximo divisor comum como sendo o número 1. Começar um laço com *i* → 2 até o menor entre *a* e *b*. A cada passo verifica se *i* é divisor de ambos, se for atualiza o máximo divisor comum para *i*. Retorna o máximo divisor comum.
  3. Algoritmo de Euclides. (procure na Internet)

Escolha uma das ideias mostradas, escreva um algoritmo em pseudo-código, implemente em C++ e teste.

1. Seja *f*1 a fração *a* e *f*2 a fração *c* . Escreva um programa que receba 4 números inteiros *a, b, c, d*, que representam

*b d*

as frações *f*1 e *f*2, calcule e mostre a fração *f*3 na forma simplificada. A soma de frações é dada por:

1. *c* (*a* × (*mmc*(*b, d*) ÷ *b*)) + (*c* × (*mmc*(*b, d*) ÷ *d*))  +  =
2. *d mmc*(*b, d*)

onde *mmc*(*b, d*) é o mínimo múltiplo comum entre *b* e *d*. Considerando *m* = *mmc*(*b, d*), temos que:

* 1. *c* (*a* × *m* ÷ *b*) + (*c* × *m* ÷ *d*)  +  =
  2. *d m*

Para simplificar uma fração devemos calcular o *mdc* entre o numerador e o denominador e dividir os dois pelo mdc:

* + 1. *a* ÷ *mdc*(*a, b*)

 →

* + 1. *b* ÷ *mdc*(*a, b*)

Escreva testes para o seu programa.

**Dica:** Use uma struct para definir uma fração.

Escreva uma função que receba as duas frações e retorne a soma das duas. Segue um esborço do programa completo;

**struct** fracao{

**int** numerador;

**int** denominador;

};

fracao soma(fracao &f1, fracao &f2){

*// TODO*

}

**int** main(){ fracao f1, f2;

*// TODO*

**return** 0;

}

1. A raiz quadrada de um número *n* é um número *r* tal que *r* × *r* = *n*. Um método bastante eficiente para calcula a raiz quadrada de um número *n* é o método babilônico, que consiste em aplicar a fórmula recursiva a seguir:

*rn−*1 +  *n*

*r* = *rn−*1

*n* 2

onde *rn* é a raiz a ser calculada e *rn−*1 é o valor anterior da raiz. Um valor arbitrário deve ser escolhido para iniciar a contagem. O cálculo deve parar quando um valor de erro aceitável for observado.

*Exemplo:* Considere o cálculo da raiz de 42, que é aproximadamente 6*.*48074069840786. Vamos considerar um erro de 0*.*001 para calcular essa raiz.O primeiro valor candidato a raiz pode ser escolhido aleatoriamente para qualquer valor entre 1 e 42. Vamos escolher *r*0 como sendo 42 ÷ 2 → 21. Dessa forma temos:

|  |  |
| --- | --- |
|  | (1) |
| *.*5760 | (2) |
| ≈ 6*.*5599 | (3) |
| ≈ 6*.*4812 | (4) |
| ≈ 6*.*4807 | (5) |
| ≈ 6*.*4807 | (6) |

*r*1 =

21 + 42

2

21

= 11*.*5

11*.*5 + 42

*r*2 = 11*.*5 ≈ 7

2

42

7*.*5760 +

*r*3 = 7*.*5760

2

42

6*.*5599 +

*r*4 = 6*.*5599

2

42

6*.*4812 +

*r*5 = 6*.*4812

2

42

6*.*4807 +

*r*6 = 6*.*4807

2

Observe que após 6 passos há uma convergência, já que *r*5 e *r*6 são, aproximadamente, o mesmo valor. Considerando o erro de 0*.*001 já temos o valor da raiz em *r*5, já que |*r*5 − *r*4| é menor do que 0*.*001.

Escreva um algoritmo que calcule a raiz quadrada de um número real com erro de 0*.*001. Escreva um programa em C++ que implemente o algoritmo, usando variáveis de precisão dupla, double, com erro de 0*.*001. O programa deve mostrar o resultado com 3 casas decimais.

RESPOSTAS:

1 -

b)

|  |  |  |
| --- | --- | --- |
| **a** | **b** | **soma** |
| 10 | 20 | 30 |
| −10 | 10 | 0 |
| −2 | −3 | -5 |
| 1000000000 | 1000000000 | 2000000000 |
| −1000000001 | 1000000005 | Valores Aleatórios |
| 2000000000 | 1000000000 | -1294967296 |
| −1212121212 | −2987654321 | 486663904 |

c) O motivo dos erros dos ultimas 3 somas se da pelo limite que a tipagem INT possui, no caso, 32 Bits, que limita

d) A correção nesse caso seria utilizar uma tipagem de limite maior, neste caso, a LONG LONG, resolveria os problemas aqui dispostos!

2 –