

Flávio Augusto de Freitas
Introdução à Programação em Linguagem C/C++

<http://flavioaf.blogspot.com>

C/C++

Tutorial 15 (usando Dev-C++ versão 4.9.9.2)



2011

1 INTRODUÇÃO

Esta série de tutoriais foi escrita usando o **Microsoft Windows 7 Ultimate** e o **Bloodshed Dev-C++** versão 4.9.9.2, que pode ser baixada em <http://www.bloodshed.net>. Se alguém quiser adquirir mais conhecimentos e quiser aprofundar no assunto, visite <http://www.cplusplus.com/>.

Qualquer **exemplo de programa** nesta série de tutoriais implica **em um novo projeto**, dentro de uma **nova pasta**. Se não souber como criar uma nova pasta, volte e refaça o tutorial 1.

2 O QUE SÃO FUNÇÕES?

Se ainda não sabe, veja o tutorial 14.

2.1 SOLUCIONANDO UM PROBLEMA MATEMÁTICO

No tutorial 14 nós elaboramos o algoritmo para solucionar a equação abaixo:

$$s = \sum_{i=0}^{100} \frac{(100-i)!}{2 \times i!}$$

O somatório acima é o mesmo que $s = \frac{(100-0)!}{2 \times 0!} + \frac{(100-1)!}{2 \times 1!} + \frac{(100-2)!}{2 \times 2!} + \dots + \frac{(100-100)!}{2 \times 100!}$, onde $i = 0, 1, 2, \dots, 100$

Vimos que você precisará calcular 101 vezes (0 a 100) o fatorial de $100 - i$ e o fatorial de i , além de calcular a divisão e acumular o resultado em uma variável.

Nosso algoritmo ficou assim:

Passo 1: Zerar a variável s

Passo 2: Iniciar i com 0

Passo 3: Calcular o fatorial de $100 - i$

Passo 4: Calcular o fatorial de i

*Passo 5: Calcular $(100 - i)! / (2 * i!)$*

Passo 6: Somar o resultado do Passo 5 à variável s

Passo 7: Incrementar i

Passo 8: Voltar ao Passo 3 até que i seja igual a 100

Passo 9: Imprimir s

Ok, mas o que é um fatorial?

Fatorial é uma função matemática denominada por $n!$ (leia-se fatorial de n ou n fatorial) e definida como:

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ n \times (n-1) \times (n-2) \times \dots \times 1, & \text{se } n > 1 \end{cases}$$

A última linha da fórmula acima nos dá o algoritmo, que em C ficará parecido com:

```
int fatorial(int n) {
    int fat = 1; // 1, pois vamos multiplicar
    int i;
    for(i=1; i<=n; i++) // i = 1, 2, 3, ..., n
        fat = fat * i;
    return fat; // retorna o fatorial calculado
}
```

Entretanto há um problema com essa função. Consegue descobrir? É uma questão de escopo de variável. O problema está na declaração da função: **int fatorial(int n)**, ou seja, a função recebe um valor n do tipo inteiro e devolve também um valor inteiro. Temos de tomar cuidado com funções de cálculo, principalmente com essa de fatorial, pois os fatoriais crescem muito rapidamente. Vejamos uma lista dos primeiros fatoriais:

n	n!
0	1
1	1
5	120
7	5.040
8	40.320 > tipo int (32767)
10	3.628.800
20	2.432.902.008.176.640.000
100	9,332621544394415x10 ¹⁵⁷ 158 dígitos

Pobrezinha da nossa função, que só trabalha com números até 32767 (int), nunca chegará ao fatorial de 100 (158 algarismos). Precisamos de algo maior. Qual tipo numérico em C é capaz de trabalhar com tal grandeza?

tic tac tic tac tic tac

Procurando por um tipo numérico em C que possa trabalhar com um numeral tão grande, encontramos:

unsigned long int, que pode armazenar até 4.294.967.295, bem menor que 100!. Na verdade só chegaríamos até o fatorial de 12.

Continuemos nossa busca...

Procurando por um tipo numérico em C que possa trabalhar com um numeral maior que unsigned long int, só encontraremos tipos de ponto flutuante. Encontramos:

float, com limites entre $[3,4 \times 10^{-38}$ a $3,4 \times 10^{38}]$ e double, com limites entre $[1,7 \times 10^{-308}$ a $1,7 \times 10^{308}]$. Este último é o nosso tipo procurado. Então podemos reescrever nossa função de fatorial para:

```
double fatorial(double n) {
    double fat = 1.0; // 1, pois vamos multiplicar
    double i;
    for(i=1; i<=n; i+=1.0) // i = 1, 2, 3, ..., n
        fat = fat * i;
    return fat; // retorna o fatorial calculado
}
```

Agora sim, estamos prontos para escrever o programa que calcula o somatório $s = \sum_{i=0}^{100} \frac{(100-i)!}{2 \times i!}$:

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;
```

```
double fatorial(double n) {
    double fat = 1.0; // 1, pois vamos multiplicar
    double i;

    for(i=1; i<=n; i+=1.0) // i = 1, 2, 3, ..., n
        fat = fat * i;
    return fat; // retorna o fatorial calculado
}
```

```
int main(int argc, char *argv[])
{
    double somatorio;
    double i;
    double fat1, fat2;

    somatorio = 0.0;
    for(i=0.0; i<=100.0; i+=1.0) {
        fat1 = fatorial(100 - i);
        fat2 = fatorial(i);
        somatorio += fat1 / (2 * fat2);
    }
    printf("Somatorio = %lf\n\n", somatorio);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Pronto! Não se assuste com o resultado. Lembra? Fatoriais ficam muito grandes muito

rapidamente, então o resultado será **monstruoso**.

3 EXERCÍCIOS RESOLVIDOS

1. Escreva um programa usando função para calcular uma expressão numérica do tipo $a + b$.

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;
```

```
float soma(float a, float b) {
    return a + b;
}
```

```
int main(int argc, char *argv[])
{
```

```
    float n1, n2;
```

```
    printf("Numero 1: ");
    scanf("%f", &n1);
    printf("\n\n");
```

```
    printf("Numero 2: ");
    scanf("%f", &n2);
    printf("\n\n");
```

```
    printf("Soma de %f e %f = %f\n\n", n1, n2,
    soma(n1, n2));
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

2. Escreva um programa usando função para calcular uma expressão numérica do tipo $a - b$.

```
#include <cstdlib>
#include <iostream>

using namespace std;

float diminui(float a, float b) {
    return a - b;
}

int main(int argc, char *argv[])
{
    float n1, n2;

    printf("Numero 1: ");
    scanf("%f", &n1);
    printf("\n\n");

    printf("Numero 2: ");
    scanf("%f", &n2);
    printf("\n\n");

    printf("Subtracao de %f por %f = %f\n\n",
n1, n2, diminui(n1, n2));

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3. Escreva um programa usando função para calcular uma expressão numérica do tipo $a * b$.

```
#include <cstdlib>
#include <iostream>

using namespace std;

float multiplica(float a, float b) {
    return a * b;
}

int main(int argc, char *argv[])
{
    float n1, n2;

    printf("Numero 1: ");
    scanf("%f", &n1);
    printf("\n\n");

    printf("Numero 2: ");
    scanf("%f", &n2);
    printf("\n\n");

    printf("Produto de %f por %f = %f\n\n",
n1, n2, multiplica(n1, n2));

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

4. Escreva um programa usando função para calcular uma expressão numérica do tipo a/b .

```
#include <cstdlib>
#include <iostream>

using namespace std;

float divide(float a, float b) {
    if(b == 0) {
        printf("Divisao por zero!\n\n");
        return 1e999;
    }
    else return a / b;
}

int main(int argc, char *argv[])
{
    float n1, n2;

    printf("Numero 1: ");
    scanf("%f", &n1);
    printf("\n\n");

    printf("Numero 2: ");
    scanf("%f", &n2);
    printf("\n\n");

    printf("Divisao de %f por %f = %f\n\n", n1,
n2, divide(n1, n2));

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

4 SOLUÇÃO DOS EXERCÍCIOS PROPOSTOS DO TUTORIAL 14

- a) Escreva uma função que receba um vetor de inteiros não ordenados e um valor inteiro que será pesquisado no vetor. A função deve retornar o elemento do vetor que está mais próximo do valor inteiro. É possível que haja mais de um elemento que esteja com igual proximidade do elemento pesquisado. Se for necessário, a função pode receber também o tamanho do vetor como parâmetro (válido para implementações em C).

```
#include <cstdlib>
#include <iostream>
#include <math.h>

using namespace std;
const int MAXTAM = 10;

int busca_proximo(float val, float v[]) {
    int i, p;
    float dif;

    dif = fabs(v[0] - val);
    p = 0;
    for(i=1; i<MAXTAM; i++) {
        printf("vet[%d]= %f > %f; ", i, v[i], fabs(v[i]
- val));
        if(fabs(v[i] - val) < dif) {
            dif = fabs(v[i] - val);
            p = i;
        }
    }
    return p;
}

int main(int argc, char *argv[])
{
    float vet[MAXTAM] = {0, 1, 2, 3, 4. 4, 4.5, 6,
7, 8, 9};
    float valor;
    int i, p;

    printf("Informe o valor procurado (4.47 ou
4.41, por exemplo): ");
    scanf("%f", &valor);
    printf("\n\n");
    i = busca_proximo(valor, vet);
    printf("\n\nValor encontrado vet[%d]=
%f\n\n", i, vet[i]);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Outra solução, mas desta vez lendo o vetor:

```
#include <cstdlib>
#include <iostream>

using namespace std;

const int MAX = 10;

int main(int argc, char *argv[])
{
    int vet[MAX];
    int i;
    int valor;
    int diff;

    for(i=0;i<MAX;i++) {
        printf("vet[%d]= ",i+1);
        scanf("%d", &vet[i]);
        putchar('\n');
    }
    printf("\n\nValor procurado: ");
    scanf("%d", &valor);
    putchar('\n');

    // encontra menor diferença
    diff = abs(vet[0] - valor);
    for(i=1;i<MAX;i++)
        if(abs(vet[i]-valor)<diff)
            diff = abs(vet[i]-valor);

    // mostra vetor
    for(i=0;i<MAX;i++)
        printf("%2d: %2d ", i+1, vet[i]);
    printf("\n\n");

    // procura valor
    for(i=0;i<MAX;i++)
        if(abs(vet[i]-valor)==diff)
            printf( "___^__ ");
        else printf("    ");
    printf("\n\n");

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Se o vetor estivesse ordenado, haveria diferença na implementação?

Sim.

Haveria alguma vantagem? Reflita sobre isso.
Se o vetor estivesse em ordem crescente, uma simplificação poderia ser feita, pois, encontrado o valor com menor diferença para o valor procurado,

poderíamos parar a busca, já que qualquer valor posterior teria diferença maior ainda.

- b) Escreva uma função que leia um vetor de inteiros ordenados de forma **não decrescente** e que imprima somente os números que não sejam repetidos.

```
#include <cstdlib>
#include <iostream>

using namespace std;

const int MAX = 10;

int qtdRepeticoes(int v[], int n);

int main(int argc, char *argv[])
{
    int vet[MAX];
    int i;

    for(i=0;i<MAX;i++) {
        printf("vet[%d]= ", i+1);
        scanf("%d", &vet[i]);
        putchar('\n');
    }
    printf("\n\n");

    printf("Vetor: todos os elementos\n");
    for(i=0;i<MAX;i++)
        printf("%d ", vet[i]);
    putchar('\n');

    printf("Vetor: nao repetidos\n");
    for(i=0;i<MAX;i++)
        if(qtdRepeticoes(vet, vet[i])<2)
            printf("%d ", vet[i]);
    printf("\n\n");

    system("PAUSE");
    return EXIT_SUCCESS;
}

int qtdRepeticoes(int v[], int n) {
    int rep = 0;
    int i;

    for(i=0;i<MAX;i++)
        if(v[i]==n) rep++;
    return rep;
}
```

- c) Faça uma função para ler um vetor. Este procedimento deve receber o número de elementos do vetor e retornar o vetor lido. Faça também um procedimento para mostrar os elementos de um vetor. Este procedimento deve receber o vetor e o número de elementos deste vetor. Faça um algoritmo e um programa que leia 2 vetores A (com 5 elementos) e B (com 5 elementos) utilizando o procedimento de leitura de vetor. O algoritmo/programa deverá fazer com que o vetor C receba os elementos do vetor A multiplicados pelos elementos correspondentes do vetor B. Por fim o algoritmo/programa deverá chamar o procedimento que mostra os elementos de um vetor para mostrar os elementos dos vetores A, B e C.

```
#include <cstdlib>
#include <iostream>

using namespace std;

const int MAX_A = 5;
const int MAX_B = 5;
const int MAX_C = 5;

void leVetor(int v[], int t);
void imprimeVetor(int v[], int t);
void multiplicaVetores(int v1[], int v2[], int v3[], int t);

int main(int argc, char *argv[])
{
    int A[MAX_A];
    int B[MAX_B];
    int C[MAX_C];

    printf("Vetor A\n");
    leVetor(A, MAX_A);
    printf("\nVetor B\n");
    leVetor(B, MAX_B);
    multiplicaVetores(A, B, C, MAX_C);
    printf("\nVetor A\n");
    imprimeVetor(A, MAX_A);
    printf("\nVetor B\n");
    imprimeVetor(B, MAX_B);
    printf("\nVetor C\n");
    imprimeVetor(C, MAX_C);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
void leVetor(int v[], int t) {
    int i;

    printf("\nInforme valores do vetor\n");
    for(i=0;i<t;i++) {
        printf("%d: ", i+1);
        scanf("%d", &v[i]);
        putchar('\n');
    }
}

void imprimeVetor(int v[], int t) {
    int i;

    printf("\nValores do vetor\n");
    for(i=0;i<t;i++) {
        printf("%d: %d ", i+1, v[i]);
    }
    putchar('\n');
}

void multiplicaVetores(int v1[], int v2[], int v3[], int t) {
    int i;

    for(i=0;i<t;i++)
        v3[i] = v1[i] * v2[i];
}
```

- d) Escreva uma função `int remove_dup(float v[], int n)` receba um vetor e verifique a existência de elementos duplicados. Caso não existam elementos duplicados retorne zero. Caso existam, remova estes elementos (deixando apenas um) e retorne o número de elementos removidos.

```
#include <cstdlib>
#include <iostream>

using namespace std;

void imprimeVetor(float v[], int n);
void leVetor(float v[], int n);
int remove_dup(float v[], int n);
void ordenaVetor(float v[], int n);

int MAX = 10;

int main(int argc, char *argv[])
{
    int dup;
    float vet[MAX];

    leVetor(vet, MAX);
    ordenaVetor(vet, MAX);
    imprimeVetor(vet, MAX);
    if(!(dup=remove_dup(vet, MAX)))
        printf("Nao ha valores duplicados...\n\n");
    else
    {
        printf("Encontrados %d duplicatas. Duplicatas removidas...\n", dup);
        imprimeVetor(vet, MAX);
    }

    system("PAUSE");
    return EXIT_SUCCESS;
}

void leVetor(float v[], int n) {
    int i;

    printf("Lendo %d valores...\n", n);
    for(i=0;i<n;i++) {
        printf("vetor[%d]= ", i+1);
        scanf("%f", &v[i]);
        //putchar('\n');
    }
}
```

```
void imprimeVetor(float v[], int n) {
    int i;

    printf("Imprimindo %d valores...\n", n);
    for(i=0;i<n;i++)
        printf("%2d: %0.3f ", i+1, v[i]);
    putchar('\n');
}

int remove_dup(float v[], int n) {
    float temp;
    int i;
    int dup = 0;
    int tam;
    float v_temp[n]; // vetor temporario

    tam = n;
    temp = v[0]; // guarda ultimo valor lido
    v_temp[0] = temp; // guarda valor nao repetido

    // guarda somente valores nao repetidos em v_temp
    for(i=1;i<n;i++) {
        if(v[i]!=temp) { // valor atual é diferente do guardado
            v_temp[i]=v[i]; // transfere valor para v_temp
            temp=v[i]; // atualiza temp
        }
        else
        {
            temp=v[i]; // atualiza temp
            tam--; // v_temp ficará menor
            dup++; // quantidade de duplicatas aumenta
        }
    }

    // atualiza vetor sem as repetições
    MAX = tam;
    for(i=0;i<n;i++)
        v[i]=v_temp[i];

    return dup;
}
```



```

void ordenaVetor(float v[], int n) {
    float temp;
    int i;
    int j;

    printf("Ordenando vetor...\n", n);
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
            if(v[i]>v[j]) {
                temp = v[i];
                v[i] = v[j];
                v[j] = temp;
            }
    printf("Vetor ordenado...\n", n);
}

```

- e) Escreva uma função void insert(float **v**[], int n, float **valor**, int **pos**) que faça a inserção de **valor** na posição **pos** do vetor **v**, deslocando os demais elementos.

```

#include <cstdlib>
#include <iostream>

using namespace std;

void imprimeVetor(float v[], int n);
void leVetor(float v[], int n);
void insert(float v[], int n, int valor, int pos);

const int MAXTAM = 10;
int TAM = 5;

int main(int argc, char *argv[])
{
    int dup;
    float vet[MAXTAM];

    leVetor(vet, TAM);
    imprimeVetor(vet, TAM);
    putchar('\n');
    insert(vet, TAM, -10, 3);
    imprimeVetor(vet, TAM);
    putchar('\n');
    insert(vet, TAM, -20, 1);
    imprimeVetor(vet, TAM);
    putchar('\n');

    system("PAUSE");
    return EXIT_SUCCESS;
}

```

```

void leVetor(float v[], int n) {
    int i;

    printf("Lendo %d valores...\n", n);
    for(i=0;i<n;i++) {
        printf("vetor[%d]= ", i);
        scanf("%f", &v[i]);
    }
}

void imprimeVetor(float v[], int n) {
    int i;

    printf("Imprimindo %d valores...\n", n);
    for(i=0;i<n;i++)
        printf("%2d: %0.3f ", i, v[i]);
    putchar('\n');
}

void insert(float v[], int n, int valor, int pos) {
    int i;

    // desloca valores
    for(i=n;i>pos;i--)
        v[i]=v[i-1];

    // insere valor
    v[pos]=valor;

    TAM += 1;

    printf("Inserido valor %d na posicao %d...\n", valor, pos);
}

```

5 EXERCÍCIOS PROPOSTOS

- a) Escreva um programa usando função para calcular uma expressão numérica passada na forma *número1 operador número2*, onde operador pode ser qualquer um de: +, -, *, /, que o usuário fornecer. Os números também são fornecidos pelo usuário.
- b) Faça uma função que verifique se um número inteiro é perfeito ou não. Um número inteiro é dito perfeito quando ele é igual a soma dos seus divisores excetuando-se ele próprio. (Exemplo: 6 é perfeito, $6 = 1 + 2 + 3$, que são seus divisores, exceto o próprio 6). A função deve retornar um valor booleano. Use a função para testar os primeiros 200 números naturais.
- c) Faça uma função que recebe um valor inteiro e verifica se o valor é positivo ou negativo. A função deve retornar um valor booleano.
- d) Escreva uma função que recebe por parâmetro um valor positivo N e retorna o valor de S.
$$S = 1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/N.$$
- e) Escreva uma função que recebe por parâmetro um valor inteiro e positivo N e retorna o valor de S.
$$S = 1 + 1/1! + 1/2! + 1/3! + 1/N!$$
- f) Escreva uma função que recebe por parâmetro um valor inteiro e positivo N e retorna o valor de S.
$$S = 3/4 + 5/5 + 7/6 + \dots + (2n + 1) / (n + 3)$$

6 TERMINAMOS

Terminamos por aqui. O que está esperando, saia do Dev-C++ e corra para pegar o próximo tutorial em <http://flavioaf.blogspot.com>. Siga o blog, assim você fica sabendo das novidades no momento em que forem publicadas. Seguindo o blog você se mantém sempre atualizado de qualquer lançamento novo.