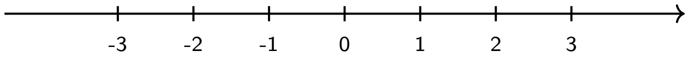
Programando com números inteiros
Professor Wladimir

Programando com números inteiros

Os números inteiros podem ser entendidos como uma lista



Então podemos continuar usando a nossa metáfora do dedo que anda de um lado para outro.

Somatório 1 até N

Imagine que nós temos um número N. Queremos calcular:

$$S = 1 + 2 + 3 + \ldots + N \tag{1}$$

Por exemplo, para N = 5, temos

$$S = 1 + 2 + 3 + 4 + 5 = 15 \tag{2}$$

Esse cálculo pode ser feito assim:

```c
#include <stdio.h>

int N = 5;
int S;

int main(){
    int i;
    S = 0;
    for(i=1;i<=N;i++)
        S = S + i;
}
```

Somatório dos quadrados de 1 até N

Imagine que nós temos um número N. Queremos calcular:

$$S = 1^2 + 2^2 + 3^2 + \ldots + N^2 \tag{3}$$

Por exemplo, para N = 5, temos

$$S = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55 \tag{4}$$

Esse cálculo pode ser feito assim:

```c
#include <stdio.h>
int N = 5;
int S;
int main(){
    int i;
    S = 0;
    for(i=1;i<=N;i++)
        S = S + i*i;
}
```

Somatório dos multiplos de 3 ou 5 de 1 até N

Imagine que nós temos um número N. Queremos calcular:

$$S = 3 + 5 + 6 + 9 + \ldots + N \tag{5}$$

Por exemplo, para N = 10, temos

$$S = 3 + 5 + 6 + 9 = 23 \tag{6}$$

Esse cálculo pode ser feito assim:

```c
#include <stdio.h>
int N = 5;
int S;
int main(){
    int i;
    S = 0;
    for(i=1;i<=N;i++){
        if(i%3==0 || i%5==0)
            S = S + i;
    }
}
```

ou podemos fazer de maneira esperta:

Quadrado perfeito

O número N^2 pode ser escrito como a soma dos N primeiros ímpares.

- $1^2 = 1$
- $2^2 = 1 + 3$
- $3^2 = 1 + 3 + 5$
- $4^2 = 1 + 3 + 5 + 7$

Podemos calcular N^2 usando apenas soma da seguinte maneira:

```c
#include <stdio.h>
int N = 10;
int resp;
int main(){
    int i , ni;
    resp = 0;
    ni   = 1;
    for(i=1;i<=N;i++){
        S = S + ni;
        ni = ni + 2;
    }
}
```

Ou podemos usar o seguinte truque i-ésimo número ímpar é igual a:

$$2 * i - 1 \tag{7}$$

```c
#include <stdio.h>
int N = 10;
int resp;
int main(){
    int i , ni;
    resp = 0;
    ni   = 1;
    for(i=1;i<=N;i++){
        S = S + 2*i - 1
    }
}
```

ou ainda

```c
#include <stdio.h>
int N = 10;
int resp;
int main(){
    int i , ni;
    resp = 0;
    for(i=1, ni = 1;i<=N;i++, ni = ni +2){
        S = S + ni;
    }
}
```

Estimando a raiz quadrada

Agora que a gente já sabe calcular os quadrados perfeitos, não é difícil estimar a raiz quadrada de um número.

Quer dizer, a ideia é simples.

Imagine que a gente quer estimar a raiz quadrada de 39.

Daí, a gente calcula todos os quadrados perfeitos até passar de 39

1	2	3	4	5	6	7
1	4	9	16	25	36	63

↑
39

e descobre que a sua raiz quadrada é alguma coisa entre 6 e 7.

ou seja, queremos encontrar o maior k tal que $k^2 \leq N$

```
#include <stdio.h>

int N = 0;

int main(){
    int k = 0;
    int Q = 0; // Q = k^2
    int ni = 1;

    while(Q+ni <= N){
        k++;
        Q = Q + ni;
        ni = ni + 2;
    }

    printf("a raiz quadrada de %d e igual a %d\n", N, k);

}
```

Contando pares ordenados

Dado um número inteiro N, determine quantos pares ordenados (x, y) existem tais que:

$$0 \leq x, y \leq N \text{ e } 3|x + y$$

Ou seja, conte quantos pares de inteiros distintos podem ser formados onde x é estritamente menor que y, e ambos estão no intervalo de 0 a N (inclusive).

Por exemplo, para N = 3, temos os 5 pares:
(0,0),(0,3)
(1,2)
(2,1)
(3,0), (3,3)

```c
#include <stdio.h>

int N = 3;

int conta_pares(int N){
    int cont = 0;
    int x, y;
    for(x = 0; x <= N; x++){
        for(y = 0; y <= N; y++){
            if( (x+y)%3 == 0){
                printf("(%d, %d)\n", x,y);
                cont++;
            }
        }
    }
}
int main(){
    int cont = conta_pares(N);
}
```

Número primo

Os números primos são os números que são divisíveis por 1 ou por eles mesmos.

Imagine que nós temos um número N e queremos a variável primo indique se o número é primo ou não.

Vamos procurar algum divisor d entre $2, 3, \ldots, N-1$

```c
#include <stdio.h>
int N = 10;
int primo;
int main(){
    int d;
    primo = 1;
    for(d=2;d<=N-1;d++){
        if(N%d==0){
            primo = 0;
            break;
        }
    }
}
```

Criando uma caixinha para primo:

```c
#include <stdio.h>
int N = 10;
int primo(int N){
    int d;
    for(d=2;d<=N-1;d++){
        if(N%d==0){
            return 0;
        }
    }
    return 1;
}
```

Números primos (versão 2.0)

A gente já viu como verificar se um número é primo ou não.

Mas agora, a gente pode ser um pouquinho mais esperto.

Quer dizer, não é preciso examinar todos os números entre 2 e $N-1$ para encontrar um divisor de k.

Porque, se não existe divisor entre 2 e $\sqrt{N}$, então não existe divisor nenhum — (*porque?*)

```c
#include <stdio.h>
int N = 10;
int primo(int N){
    int d;
    for(d=2;d*d<=N;d++){
        if(N%d==0){
            return 0;
        }
    }
    return 1;
}
```

```c
#include <stdio.h>
int N = 10;
int primo(int N){
    int d;
    if(N == 2) return 1;
    if(N%2 == 0) return 0;
    for(d=3;d*d<=N;d+=2){
        if(N%d==0){
            return 0;
        }
    }
    return 1;
}
```

Conta dígitos

Sabemos que os números entre 0 e 9 possui apenas 1 dígitos. Para os números maiores que 10, podemos dividir por 10 para remover o último dígitos. Logo, o algoritmo para contagem dos dígitos pode ser feito da seguinte maneira:

```c
#include <stdio.h>
int N = 10;
int conta_digitos(int N){
    int cont = 0;
    while(N > 10){
        cont++;
        N = N/10;
    }
    cont++;
    return cont;
}

int main(){
    int cont = conta_digitos(321);
}
```

Soma dígitos

```c
#include <stdio.h>
int N = 10;
int soma_digitos(int N){
    int soma = 0;
    while(N > 10){
        soma += N%10;
        N = N/10;
    }
    soma += N;
    return soma;
}

int main(){
    int cont = soma_digitos(321);
}
```

Potência

Dado um número inteiro a e um inteiro n, o objetivo é calcular o valor de a^n, ou seja, a potência de base a e expoente n.

Uma forma simples de resolver esse problema é utilizando um método incremental, em que calculamos sucessivamente as potências de a desde a^0 até a^n. Para isso, podemos usar um vetor auxiliar para armazenar os resultados intermediários e evitar cálculos repetidos. A fórmula utilizada é:

$$a^i = a^{i-1} \cdot a$$

para todo i de 1 até n, sendo que $a^0 = 1$.

A seguir, apresentamos uma implementação em C++ desse método:

```
#include <stdio.h>
int potencia(int a, int n){
    int i;
    int pot[n+1];
    pot[0] = 1;
    for(i = 1; i <= n; i++){
        pot[i] = pot[i-1]*a;
    }
    return pot[n];
}


int main(){
    int pot = potencia(2, 5);
}
```

Uma forma mais eficiente em termos de uso de memória para calcular a potência a^n é evitar o uso de um vetor auxiliar e acumular o resultado diretamente em uma única variável. Essa abordagem economiza espaço, pois só precisamos manter o valor atual da potência durante o cálculo.

A cada iteração do laço, multiplicamos o resultado acumulado pela base a, conforme a regra:

$$\text{potência atual} = \text{potência anterior} \times a$$

```
#include <stdio.h>
int potencia(int a, int n){
    int i;
    int pot
    pot = 1;
    for(i = 1; i <= n; i++){
        pot = pot*a;
    }
    return pot;
}


int main(){
    int pot = potencia(2, 5);
}
```

Avaliação de um polinômio

Seja um polinômio $P(x)$ definido pelos coeficientes $a = [a_0, a_1, \ldots, a_n]$, onde a_i representa o coeficiente do termo x^i. O valor de $P(x)$ para um dado número inteiro x é calculado como:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

Ou, de forma equivalente:

$$P(x) = \sum_{i=0}^{n} a_i \cdot x^i$$

A seguir, temos uma implementação em C que realiza essa avaliação. A função ‘potencia‘ calcula x^i de maneira simples por multiplicações sucessivas, e a função ‘polinomio‘ utiliza essa potência para calcular cada termo do somatório.

```c
#include <stdio.h>
int potencia(int a, int n){
    int i;
    int pot;
    pot = 1;
    for(i = 1; i <= n; i++){
        pot = pot*a;
    }
    return pot;
}

int polinomio(int a[], int n, int x){
    int i;
    int res;
    for(i = 0; i < n; i++){
        res = res +  a[i]*potencia(x,i);
    }
    return res;
}


int main(){
    int a[3] = {6,-5,1};
    int res;
    //P(x) = x^2 - 5x + 6
    //P(2) = 4 -10 + 6 = 0
    res = polinomio(a, 3, 2);
    printf("res %d\n", res);
    //P(2) = 16 -20 + 6 = 2

    res = polinomio(a, 3, 4);
    printf("res %d\n", res);

}
```

Neste caso, a utilização de uma função separada para potencia representa uma perda de eficiência. uma maneira esperta é fazer da seguinte maneira:

```c
#include <stdio.h>
int polinomio(int a[], int n, int x){
    int i;
    int res;
    int pot = 1;
    for(i = 0; i < n; i++){
        res = res +  a[i]*pot;
        pot = pot*x;
    }
    return res;
}

int main(){
    int a[3] = {6,-5,1};
    int res;
    //P(x) = x^2 - 5x + 6
    //P(2) = 4 -10 + 6 = 0
    res = polinomio(a, 3, 2);
    printf("res %d\n", res);
    //P(2) = 16 -20 + 6 = 2

    res = polinomio(a, 3, 4);
    printf("res %d\n", res);

}
```

Fatoração

```c
#include <stdio.h>

int main(){
    int N;
    int p = 2;

    scanf("%d", &N);

    printf("N = ");
    while(N > 1){
        //printf("testando %d\n", p);
        int exp = 0;
        while( N%p == 0){
            N = N/p;
            exp++;
        }

        if(exp>0) printf("%d^%d ", p, exp);
        p++;
    }
}

```

```
258
N = 2^1 3^1 43^1
```

Raiz Quadrada usando busca binária

```c
#include <stdio.h>

int main(){
    int N;
    scanf("%d", &N);
    int resp;
    int inicio = 0;
    int fim    = N;

    while( inicio <= fim ){
        printf("[%d, %d]\n", inicio, fim);
        int meio = (inicio+fim)/2;

        int q = meio*meio;
        if( q == N){
            resp = N;
            break;
        }else if(q < N){
            resp = meio;
            inicio = meio+1;
        }else{
            fim = meio - 1;
        }
    }

    printf("o maior k tal que k*k <=N é: %d\n", resp );

}
/*
36
[0, 36]
[0, 17]
[0, 7]
[4, 7]
[6, 7]
o maior k tal que k*k <=N é: 36
*/
```