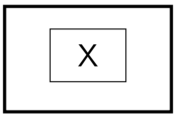
Fundamentos de programação

aula 03: Programando com números

1 Introdução

Imagine que nós queremos saber se um certo número `X` é primo ou não



Bom, um número é primo se ele só tem o número 1 e ele mesmo como divisores.

E isso é algo que um programa pode testar.

Quer dizer, na aula passada, nós vimos que o comando

```
X % Y
```

nos dá o resto da divisão de `X` por `Y`.

Daí que, se esse resto é igual a zero, nós descobrimos que `Y` é um divisor de `X`.

A ideia para o programa que verifica se `X` é primo ou não consiste em examinar todos os números entre 2 e `X − 1`, e verificar se algum deles é um divisor de `X`.

Essa é uma tarefa que pode ser realizada com o comando `Para`

```
Para  Y <-- 2  Até  X-1  Faça

   Se ( X % Y == 0 )
```

Certo.

Para completar o programa, nós vamos utilizar uma variável auxiliar `Aux`, que vai indicar se algum divisor foi encontrado ou não.

Quer dizer, no início do programa `Aux` vale 0.

E, no momento em que nós encontramos um divisor de `X`, nós fazemos `Aux` receber o valor 1.

Daí, no final do programa, basta testar se `Aux` é igual a 0 ou não.

Abaixo nós temos o programa completo

```
Aux  <--  0
Para  Y <-- 2  Até  X-1  Faça
   Se ( X % Y == 0 )
      Aux  <--  1
Se ( Aux = 0 )
Então  Print ("X é primo")
Senão  Print ("X não é primo")
```

E abaixo nós temos a versão desse programa na linguagem C.

```
#include <stdlib.h>
#include <stdio.h>

int  X = 427;
int  Y, Aux = 0;

int main()
 {
    for (Y=2; Y<X-1; Y++)
       if ( X % Y == 0 )   Aux = 1;

    if ( Aux == 0 )   printf("X é primo");
    else              printf("X não é primo");
 }
```

Esse programa tem uma pequena novidade

- A instrução que incrementa 1 ao valor de uma variável

  ```
  Y = Y + 1
  ```

  é tão comum, que a linguagem C nos dá a seguinte abreviação para ela

  ```
  Y ++
  ```

  Essa abreviação foi usada no comando `for`.

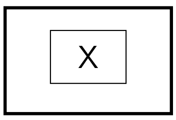
  Além disso, nós também temos a abreviação

  ```
  Y --
  ```

  que decrementa 1 do valor de `Y`.

2 Programando com números

Agora imagine que nós queremos saber se `X` é um quadrado perfeito ou não



Bom, `X` é um quadrado perfeito se existe algum número `Y` tal que

$$\mathtt{Y}^2 = \mathtt{X}$$

Além disso, se `X` é um número inteiro (positivo), então esse número `Y` deve ser menor ou igual a `X`.

Daí que, nós podemos procurar esse `Y` (se é que ele existe) usando o comando `Para`.

Abaixo nós temos o programa que implementa essa ideia

```
Aux  <--  0
Para  Y <-- 1  Até  X  Faça
   Se ( Y * Y == X )
      Aux  <--  1
Se ( Aux = 1 )
Então  Print ("X é um quadrado perfeito")
Senão  Print ("X não é um quadrado perfeito")
```

Note que na linha

```
Se ( Y * Y == X )
```

nós estamos fazendo uma conta e testando se o seu resultado é igual a `X`.

Mas, pode fazer isso?

Sim, pode.

Abaixo nós temos a versão do programa na linguagem C

```
#include <stdlib.h>
#include <stdio.h>

int  X = 329, Y, Aux = 0;

int main()
 {
    for (Y=0; Y<=X; Y++)
       if ( Y * Y == X )   Aux = 1;

    if ( Aux == 0 )   printf("X é um quadrado perfeito");
    else              printf("X não é um quadrado perfeito");
 }
```

A seguir, nós vamos ver outros exemplos de programas com números.

Exemplos

a. **Uma pequena esperteza**

Considere o número

$$418$$

Esse número não é um quadrado perfeito.

Quer dizer, o quadrado perfeito mais próximo é

$$20^2 \quad = \quad 400$$

E o quadrado perfeito que vem a seguir

$$21^2 \quad = \quad 442$$

é maior 418.

Daí que, no momento em que nós chegamos a `Y` $= 21$, nós já temos a certeza de que `X` $= 418$ não é um quadrado perfeito.

Porque a partir daí, nós sempre vamos ter que $\mathtt{Y}^2$ é maior que `X`.

Mas, o nosso programa continua testando todos esses valores de `Y`.

A boa notícia é que a linguagem C possui um comando que permite interromper a repetição antes da hora: o *comando* `break`.

A coisa funciona assim

```
      #include <stdlib.h>
      #include <stdio.h>

      int  X = 329;
      int  Y, Aux = 0;

      int main()
       {
          for (Y=0; Y<=X; Y++)
           {
             if ( Y * Y == X )   Aux = 1;

   =>        if ( Y * Y > X )  break;
           }

          if ( Aux == 0 )   printf("X é um quadrado perfeito");
          else              printf("X não é um quadrado perfeito");

          printf ("Y termina com o valor %d", Y);
       }
```

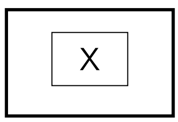
Quer dizer, no momento em que `Y` alcança o valor 21, o programa executa a instrução `break`, e o comando `for` é interrompido.

Nós colocamos um `printf` no final do programa para verificar que `Y` realmente termina a execução com o valor 21 — (*experimente para ver*).

◇

b. **Outra pequena esperteza**

Considere outra vez o problema de determinar se `X` é primo ou não



Nós já sabemos que `X` é primo se ele não possui nenhum divisor diferente de 1 e dele mesmo.

Agora, nós acrescentamos a observação de que se esse divisor existe, então ele deve ser algum número entre 2 e $\sqrt{\mathtt{X}}$.

(*Porque?*)

Mas isso significa que, para saber se `X` é primo ou não, basta testar os números dentro dessa faixa.

Para fazer isso, nós precisamos calcular o valor de $\sqrt{\mathtt{X}}$.

Mas isso nós já sabemos fazer.

Ou, pelo menos, nós temos uma boa aproximação — (i.e., o primeiro `Y` tal que $\mathtt{Y}^2 > \mathtt{X}$).

Então, juntando os dois programas que nós fizemos hoje, nós obtemos o seguinte

```
#include <stdlib.h>
#include <stdio.h>

int  X = 927;
int  Y, Z, Aux = 0;

int main()
 {

    for (Y=0; Y<=X; Y++)
       if ( Y * Y > X )   break;

    for (Z=2; Z<Y; Z++)
       if ( X % Z == 0 )   Aux = 1;

    if ( Aux == 0 )   printf("X é primo");
    else              printf("X não é primo");
 }
```

Quer dizer, o primeiro comando `for` obtém uma aproximação `Y` para o valor de $\sqrt{\mathtt{X}}$.

E o segundo comando `for` procura por um divisor de `X` no intervalo $[2, \mathtt{Y}]$.

Não é legal?

◇

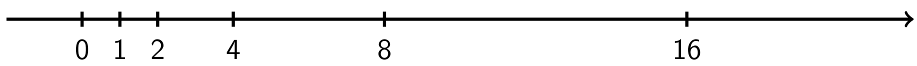
c. **Mais uma esperteza** (opcional)

Quando nós estamos trabalhando com números muito, muito grandes, toda esperteza conta para fazer o programa executar mais rápido.

A esperteza dessa vez consiste em obter a aproximação para $\sqrt{\mathtt{X}}$ mais rápido.

E a ideia consiste em pular os números pequenos para chegar logo próximo de $\sqrt{\mathtt{X}}$.

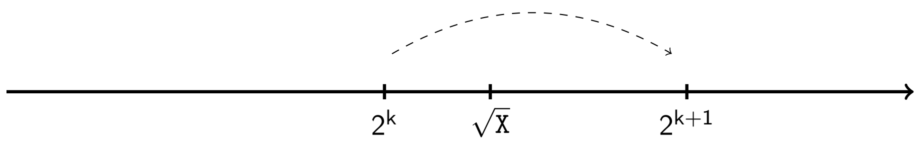
Uma maneira de fazer isso consiste em olhar apenas para as potências de 2



Note que os saltos que a gente dá com as potências de 2 são cada vez maiores, e é isso que faz com que a gente chegue logo perto de $\sqrt{\mathtt{X}}$.

Mas, muito provavelmente, a gente vai passar do ponto ...

Quer dizer, a gente vai saltar por cima de $\sqrt{\mathtt{X}}$.



Mas, isso não tem problema.

Quer dizer, agora nós sabemos que $\sqrt{\mathtt{X}}$ deve estar no intervalo $[2^{\mathtt{k}}, 2^{\mathtt{k}+1}]$, e nós podemos procurar por ele lá.

O programa abaixo implementa essa ideia.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int  X = 927*927;
int  k, Y, Aux = 0;

int main()
 {

    for (k=0; k<=X; k++)
       if ( pow(2,k) * pow(2,k) > X )   break;

    for (Y=pow(2,k-1); Y<= pow(2,k); Y++)
     {
       if ( Y * Y == X )   Aux = 1;

       if ( Y * Y > X )   break;

    if ( Aux == 0 )   printf("X é um quadrado perfeito");
    else              printf("X não é um quadrado perfeito");
 }
```

Nesse programa nós utilizamos a função matemática

$$\mathtt{pow}(2, \mathtt{k})$$

que nos dá o valor de 2^k — (`pow` vem da palavra *power* que significa potência em inglês).

Esse programa faz parte da biblioteca de funções matemáticas da linguagem C.

E é por isso que dessa vez nós acrescentamos a declaração

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

no início do programa.

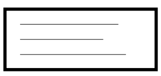
◇

d. **Todos os primos**

Imagine que nós queremos imprimir todos os números primos entre 1 e 100.

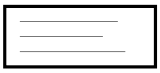
Bom, nós já temos um programa que verifica se `X` é primo ou não.

E nós podemos pensar nesse programa como uma pequena caixinha

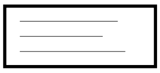


E daí, para imprimir todos os primos entre 1 e 100, nós podemos fazer o seguinte

`X = 2`

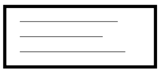


`X = 3`



...

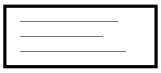
`X = 100`



Mas, nós já sabemos que essas computações repetitivas não são feitas assim.

Quer dizer, basta colocar a caixinha dentro de um comando `for`

`Para   X ← 2   Até   100   Faça`



Abaixo nós temos o programa em C que implementa essa ideia

```
#include <stdlib.h>
#include <stdio.h>

int  X, Y, Aux;

int main()
 {
    for (X=2; X<=100; X++)
     {
       Aux = 0;
       for (Y=2; Y<X-1; Y++)
          if ( X % Y == 0 )   Aux = 1;

       if ( Aux == 1 )   printf("%d ", X);
 }   }
```

◇

e. **Todos os quadrados perfeitos**

Imagine agora que nós queremos imprimir todos os quadrados perfeitos entre 0 e 100.

Então, nós poderíamos escrever simplesmente

```
for (k=0; k<=100; k++)
 {
    if ( k*k > 100)   break;
    else              printf ("%d ", k*k);
 }
```

Mas, isso não teria a menor graça ...

A seguir, nós vamos ver uma ideia diferente.

Aqui nós temos os primeiros 7 quadrados perfeitos

$$0, \quad 1, \quad 4, \quad 9, \quad 16, \quad 25, \quad 36, \quad \ldots$$

E então, você consegue ver o padrão?

Bom, o padrão aparece quando a gente calcula a diferença entre quadrados perfeitos consecutivos

$$\begin{array}{ccccccccccccc} 0, & & 1, & & 4, & & 9, & & 16, & & 25, & & 36 \\ & 1 & & 3 & & 5 & & 7 & & 9 & & 11 & \end{array}$$

É isso mesmo: a diferença entre os quadrados perfeitos consectivos são os números ímpares!

Essa observação nos permite advinhar que o próximo quadrado perfeito é

$$36 \ + \ 13 \quad = \quad 49$$

Mas, será que isso sempre funciona?

Bom, para ver isso, basta observar que

$$k^2 \quad - \quad (k-1)^2 \quad = \quad 2k \ - \ 1$$

Quer dizer,

- A diferença entre o k-ésimo e o $(k-1)$-ésimo quadrados perfeitos
  é igual a $2k-1$, que o k-ésimo número primo

Agora que nós temos essa esperteza, não é difícil colocá-la em prática no seguinte programa

◇

```
#include <stdlib.h>
#include <stdio.h>

int  Q = 0, k;

int main()
 {
    for (k=1; k<=100; k++)
     {
       printf("%d ", Q);

       Q = Q + ( 2*k + 1 );          // somando o k-ésimo ímpar

       if ( Q > 100 )  break;
 }   }
```

Não é legal?

◇

f. **O crivo de Eratósthenes**

Um dia (aprox. 300 AC), Eratósthenes teve a seguinte ideia para obter todos os números primos entre 1 e 1000

- Começando com a lista

$$2, \quad 3, \quad 4, \quad 5, \quad 6, \quad 7, \quad 8, \quad \ldots \quad \ldots, \quad 1000$$

  - primeiro eu risco todos os números múltiplos de 2 (porque nenhum deles é primo)
  - depois eu risco todos os números múltiplos de 3 (porque nenhum deles é primo)
  - e assim por diante

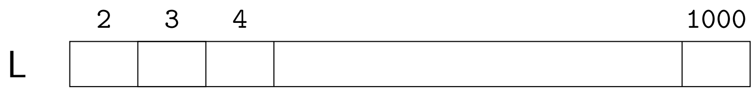
  Quer dizer, a medida que eu vou percorrendo a lista

  - sempre que eu encontro um número não riscado, eu já sei que ele é primo
  - e daí, eu risco todos os seus múltiplos (porque nenhum deles é primo)

Ora, mas isso é bem fácil de implementar em um programa de computador.

Quer dizer,

- nós vamos utilizar uma lista



  onde no início todos os elementos são zero — (i.e., o números não estão riscados).

- E daí, nós percorremos a lista, executando a lógica descrita acima.

A coisa fica assim

```
int  L[1001], i, j;

int main()
 {
    for (i=2; i<=1000; i++)    L[i] = 0;

    for (i=2; i<=1000; i++)
     {
       if ( L[i] == 0 )
        {
          for (j=2; j<=1000; j++)
           {
             if ( i * j > 1000 )   break;
             else                  L[i*j] = 1;
     }  }  }

    for (i=2; i<=1000; i++)
      if ( L[i] == 0 )  printf("%d ", L[i]);
 }
```

◇