LPG0001 – Linguagem de Programação

Estruturas de Repetição (parte 2)

Prof. Rui Jorge Tramontin Junior
Departamento de Ciência da Computação
UDESC / Joinville

Introdução

- Exemplos da aula de hoje:

1. Determinar a quantidade de divisores de um número;
2. Saber se um número é primo;
3. Geração de uma série de números primos;
4. Sequência de Fibonacci;
5. Série de Taylor para calcular a função e^x.

EXEMPLO 1: DETERMINAR A QUANTIDADE DE DIVISORES DE UM NÚMERO

Exemplo 1: determinar a quantidade de divisores de um número

- Dado um número inteiro *X*, o algoritmo faz uma variável *I* variar de 1 até *X* e testa <u>se *I* é divisor de *X*</u>;

Exemplo 1: determinar a quantidade de divisores de um número

- Dado um número inteiro *X*, o algoritmo faz uma variável *I* variar de 1 até *X* e testa se *I* é divisor de *X*;

x % i == 0

Exemplo 1: determinar a quantidade de divisores de um número

- Dado um número inteiro *X*, o algoritmo faz uma variável *I* variar de 1 até *X* e testa <u>se *I* é divisor de *X*</u>;

x % i == 0

- Cada vez que tal condição é verdadeira, incrementa-se um <u>contador</u>;

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3 → é divisor
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3 → é divisor
4
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3 → é divisor
4
5
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3 → é divisor
4
5
6
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3 → é divisor
4
5
6 → é divisor
```

Exemplo 1: determinar a quantidade de divisores de um número

- Por exemplo, para X = 6, geramos os valores de I:

```
1 → é divisor
2 → é divisor
3 → é divisor          4 divisores
4
5
6 → é divisor
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
printf("Digite um número: ");
scanf("%i", &x);
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
printf("Digite um número: ");
scanf("%i", &x);

int i, cont = 0; // zera o contador
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
printf("Digite um número: ");
scanf("%i", &x);

int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){



}
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
printf("Digite um número: ");
scanf("%i", &x);

int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){
  if( x % i == 0 ) // é divisor?


}
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
printf("Digite um número: ");
scanf("%i", &x);

int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){
  if( x % i == 0 ) // é divisor?
    cont++; // incrementa contador
}
```

Exemplo 1: determinar a quantidade de divisores de um número

```
int x;
printf("Digite um número: ");
scanf("%i", &x);

int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){
  if( x % i == 0 ) // é divisor?
    cont++; // incrementa contador
}
printf("%i tem %i divisores\n", x, cont);
```

EXEMPLO 2: DETERMINAR SE UM NÚMERO É PRIMO

Exemplo 2: determinar se um número é primo

- Uma vez que se sabe como determinar a quantidade de divisores de X, basta testar se o mesmo possui somente 2 divisores;

Exemplo 2: determinar se um número é primo

- Uma vez que se sabe como determinar a quantidade de divisores de X, basta testar se o mesmo possui somente 2 divisores;
  - X é primo, pois é divisível por 1 e por X somente!

Exemplo 2: determinar se um número é primo

- Uma vez que se sabe como determinar a quantidade de divisores de X, basta testar se o mesmo possui somente 2 divisores;
  - X é primo, pois é divisível por 1 e por X somente!

- Se for o caso, X é primo;

Exemplo 2: determinar se um número é primo

- Uma vez que se sabe como determinar a quantidade de divisores de X, basta testar se o mesmo possui somente 2 divisores;
  - X é primo, pois é divisível por 1 e por X somente!

- Se for o caso, X é primo;

- Portanto, basta incluir o teste ao final do código do exemplo anterior.

Exemplo 2: determinar se um número é primo

```
int x;
printf("Digite um número: ");
scanf("%i", &x);
int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){
  if( x % i == 0 ) // é divisor?
    cont++; // incrementa contador
}
```

Exemplo 2: determinar se um número é primo

```
int x;
printf("Digite um número: ");
scanf("%i", &x);
int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){
  if( x % i == 0 ) // é divisor?
    cont++; // incrementa contador
}
if( cont == 2 )
  printf("%i é primo\n", x);
else
  printf("%i não é primo\n", x);
```

Exemplo 2: determinar se um número é primo

```
int x;
printf("Digite um número: ");
scanf("%i", &x);
int i, cont = 0; // zera o contador
for( i = 1 ; i <= x ; i++ ){
  if( x % i == 0 ) // é divisor?
    cont++; // incrementa contador
}
if( cont == 2 )
  printf("%i é primo\n", x);
else
  printf("%i não é primo\n", x);
```

Considerações

- O algoritmo aqui apresentado é muito simples;

Considerações

- O algoritmo aqui apresentado é muito simples;

- Tende a ser ineficiente para números muito grandes.

EXEMPLO 3: GERAR OS PRIMOS DE 1 ATÉ N

Exemplo 3: gerar os primos de 1 até N

- Dado N, mostrar na tela os primos de 1 até N;

Exemplo 3: gerar os primos de 1 até N

- Dado N, mostrar na tela os primos de 1 até N;

- Deve-se usar uma estrutura de repetição (*for*) para gerar os valores de X de 1 até N;

Exemplo 3: gerar os primos de 1 até N

- Dado N, mostrar na tela os primos de 1 até N;
- Deve-se usar uma estrutura de repetição (*for*) para gerar os valores de X de 1 até N;
- Dentro dessa estrutura coloca-se o algoritmo do exemplo anterior.

Exemplo 3: gerar os primos de 1 até N

- Dado N, mostrar na tela os primos de 1 até N;
- Deve-se usar uma estrutura de repetição (*for*) para gerar os valores de X de 1 até N;
- Dentro dessa estrutura coloca-se o algoritmo do exemplo anterior.
  - Ou seja, verifica se X é primo!

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){







}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0;







}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0; // inicializado aqui!





}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0;
  for( i = 1 ; i <= x ; i++ ){



  }



}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
   int i, cont = 0;
   for( i = 1 ; i <= x ; i++ ){
      if( x % i == 0 )

   }


}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0;
  for( i = 1 ; i <= x ; i++ ){
    if( x % i == 0 )
      cont++;
  }


}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0;
  for( i = 1 ; i <= x ; i++ ){
    if( x % i == 0 )
      cont++;
  }
  if( cont == 2 )

}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0;
  for( i = 1 ; i <= x ; i++ ){
    if( x % i == 0 )
      cont++;
  }
  if( cont == 2 )
    printf("%i \n", x);
}
```

Exemplo 3: gerar os primos de 1 até N

```
int n, x;
printf("Digite um número: ");
scanf("%i", &n);
for( x = 1 ; x <= n ; x++ ){
  int i, cont = 0;
  for( i = 1 ; i <= x ; i++ ){
    if( x % i == 0 )
      cont++;
  }
  if( cont == 2 )
    printf("%i \n", x);
}
```

Considerações

- Repare que a inicialização de *cont* deve ser feita dentro do laço, para que seja zerado a cada iteração, ou seja, a cada novo valor de *x*;

Considerações

- Repare que a inicialização de *cont* deve ser feita dentro do laço, para que seja zerado a cada iteração, ou seja, a cada novo valor de *x*;

- Outro ponto que merece destaque é que o algoritmo é ineficiente e tende a ficar muito lento com N muito grande (500.000 por exemplo);

Considerações

- Repare que a inicialização de *cont* deve ser feita dentro do laço, para que seja zerado a cada iteração, ou seja, a cada novo valor de *x*;
- Outro ponto que merece destaque é que o algoritmo é ineficiente e tende a ficar muito lento com N muito grande (500.000 por exemplo);
- Em uma aula futura será mostrado como otimizar esse algoritmo.

EXEMPLO 4: SEQUÊNCIA DE FIBONACCI

Exemplo 4: Sequência de Fibonacci

- Os 2 primeiros termos da sequência são os valores 1 e 1;

Exemplo 4: Sequência de Fibonacci

- Os 2 primeiros termos da sequência são os valores 1 e 1;
- A partir do 3º termo, calcula-se o valor com base na soma dos dois anteriores;

Exemplo 4: Sequência de Fibonacci

- Os 2 primeiros termos da sequência são os valores 1 e 1;
- A partir do 3º termo, calcula-se o valor com base na soma dos dois anteriores;
- Exemplo: para N = 10, temos:

Exemplo 4: Sequência de Fibonacci

- Os 2 primeiros termos da sequência são os valores 1 e 1;

- A partir do 3º termo, calcula-se o valor com base na soma dos dois anteriores;

- Exemplo: para N = 10, temos:

Valor:	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**
Posição:	*1º*	*2º*	*3º*	*4º*	*5º*	*6º*	*7º*	*8º*	*9º*	*10º*

Exemplo 4: Sequência de Fibonacci

- A solução consiste em utilizar 3 variáveis:

Exemplo 4: Sequência de Fibonacci

- A solução consiste em utilizar 3 variáveis:
  - 2 para os valores anteriores (variáveis *a* e *b*);

Exemplo 4: Sequência de Fibonacci

- A solução consiste em utilizar 3 variáveis:
  - 2 para os valores anteriores (variáveis *a* e *b*);
  - 1 para o valor gerado (variável *atual*);

Exemplo 4: Sequência de Fibonacci

- A solução consiste em utilizar 3 variáveis:
  - 2 para os valores anteriores (variáveis *a* e *b*);
  - 1 para o valor gerado (variável *atual*);

- O algoritmo começa atribuindo-se 1 para *a* e *b*;
  - 2 primeiros termos da série;

Exemplo 4: Sequência de Fibonacci

- A solução consiste em utilizar 3 variáveis:
  - 2 para os valores anteriores (variáveis *a* e *b*);
  - 1 para o valor gerado (variável *atual*);

- O algoritmo começa atribuindo-se 1 para *a* e *b*;
  - 2 primeiros termos da série;

- O ciclo de repetições começa a partir do 3º termo:
  - Variável *atual* recebe a soma entre *a* e *b*;

Exemplo 4: Sequência de Fibonacci

- Uma vez gerado, o novo valor é mostrado na tela;

Exemplo 4: Sequência de Fibonacci

- Uma vez gerado, o novo valor é mostrado na tela;

- Em seguida, atualizamos os valores de *a* e *b* para a próxima iteração:

Exemplo 4: Sequência de Fibonacci

- Uma vez gerado, o novo valor é mostrado na tela;

- Em seguida, atualizamos os valores de *a* e *b* para a próxima iteração:

```
a = b;
```

Exemplo 4: Sequência de Fibonacci

- Uma vez gerado, o novo valor é mostrado na tela;

- Em seguida, atualizamos os valores de *a* e *b* para a próxima iteração:

```
a = b;
b = atual;
```

Exemplo 4: Sequência de Fibonacci

- Uma vez gerado, o novo valor é mostrado na tela;

- Em seguida, atualizamos os valores de *a* e *b* para a próxima iteração:

```
a = b;
b = atual;
```

- O processo se repete até que se chegue ao valor de N definida pelo usuário.

Exemplo 4: Sequência de Fibonacci

```
int n, i;
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);

int a = 1, b = 1, atual;
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);

int a = 1, b = 1, atual;

printf("1°: 1\n2°: 1\n");  // 2 primeiros termos: 1 e 1.
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);

int a = 1, b = 1, atual;

printf("1°: 1\n2°: 1\n");  // 2 primeiros termos: 1 e 1.

for ( i = 3 ; i <= n ; i++ ){




}
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);

int a = 1, b = 1, atual;

printf("1°: 1\n2°: 1\n");  // 2 primeiros termos: 1 e 1.

for ( i = 3 ; i <= n ; i++ ){
  atual = a + b;          // Gera o termo atual.




}
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);

int a = 1, b = 1, atual;

printf("1°: 1\n2°: 1\n");  // 2 primeiros termos: 1 e 1.

for ( i = 3 ; i <= n ; i++ ){
  atual = a + b;           // Gera o termo atual.
  printf("%i°: %i\n", i, atual);



}
```

Exemplo 4: Sequência de Fibonacci

```
int n, i;
printf("Digite a quantidade de termos: ");
scanf("%i", &n);

int a = 1, b = 1, atual;

printf("1°: 1\n2°: 1\n");  // 2 primeiros termos: 1 e 1.

for ( i = 3 ; i <= n ; i++ ){
  atual = a + b;            // Gera o termo atual.
  printf("%i°: %i\n", i, atual);
  a = b;                    // Atualiza a e b para a...
  b = atual;                // ...próxima iteração.
}
```

Considerações

- Este exemplo trabalha com inteiros de 32 bits;

Considerações

- Este exemplo trabalha com inteiros de 32 bits;

- Como os termos da sequência crescem rapidamente, é possível gerar uma sequência com até 46 termos;

Considerações

- Este exemplo trabalha com inteiros de 32 bits;

- Como os termos da sequência crescem rapidamente, é possível gerar uma sequência com até 46 termos;
  - A partir daí, os números são maiores do que a capacidade de representação (32 bits);

Considerações

- Este exemplo trabalha com inteiros de 32 bits;

- Como os termos da sequência crescem rapidamente, é possível gerar uma sequência com até 46 termos;
  - A partir daí, os números são maiores do que a capacidade de representação (32 bits);

- Pode-se utilizar inteiros de 64 bits (*long long int*)
  - Sequência pode ter até 92 termos.

EXEMPLO 5: SÉRIE DE TAYLOR (CÁLCULO DE E^X)

Exemplo 5: Série de Taylor (e^x)

- A série é definida da seguinte forma:

Exemplo 5: Série de Taylor (e^x)

- A série é definida da seguinte forma:

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Exemplo 5: Série de Taylor (e^x)

- A série é definida da seguinte forma:

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

- Entrada do algoritmo:
  - o valor de *x*;
  - a quantidade *n* de termos da série;

Exemplo 5: Série de Taylor (e^x)

- Quanto maior o valor de *n*, mais preciso fica o resultado;

Exemplo 5: Série de Taylor (e^x)

- Quanto maior o valor de *n*, mais preciso fica o resultado;

- É possível determinar experimentalmente quantos termos são necessários para se ter uma certa precisão (em termos de casas decimais);

Exemplo 5: Série de Taylor (e^x)

- Quanto maior o valor de *n*, mais preciso fica o resultado;

- É possível determinar experimentalmente quantos termos são necessários para se ter uma certa precisão (em termos de casas decimais);

- Uma forma simples de implementar este algoritmo é apresentada a seguir.

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série




}
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série
  float pot = 1;
  int fat = 1;




}
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série
  float pot = 1;
  int fat = 1;
  for( j = 1 ; j <= i ; j++ ){ // potenciação e fatorial


  }

}
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série
  float pot = 1;
  int fat = 1;
  for( j = 1 ; j <= i ; j++ ){ // potenciação e fatorial
    pot = pot * x;

  }

}
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série
  float pot = 1;
  int fat = 1;
  for( j = 1 ; j <= i ; j++ ){ // potenciação e fatorial
    pot = pot * x;
    fat = fat * j;
  }

}
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série
  float pot = 1;
  int fat = 1;
  for( j = 1 ; j <= i ; j++ ){ // potenciação e fatorial
    pot = pot * x;
    fat = fat * j;
  }
  e_x = e_x + pot / fat; // Acumula termo no somatório
}
```

Exemplo 5: Série de Taylor (e^x) (versão simples)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos da série
  float pot = 1;
  int fat = 1;
  for( j = 1 ; j <= i ; j++ ){ // potenciação e fatorial
    pot = pot * x;
    fat = fat * j;
  }
  e_x = e_x + pot / fat; // Acumula termo no somatório
}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

Considerações

- O *for* externo (variável *i*) gera os termos da série, de 0 até *n*;

Considerações

- O *for* externo (variável *i*) gera os termos da série, de 0 até *n*;

- O *for* interno (variável *j*) calcula, ao mesmo tempo, a *poteciação* (x^i) e o *fatorial* (i!)
  - Semelhante aos algoritmos já apresentados;

Considerações

- O *for* externo (variável *i*) gera os termos da série, de 0 até *n*;

- O *for* interno (variável *j*) calcula, ao mesmo tempo, a *poteciação* (x^i) e o *fatorial* (i!)
  - Semelhante aos algoritmos já apresentados;

- **Limitação**: este programa aceita *n* valendo no máximo 12, pois uma variável *int* não consegue guardar o valor de 13!

Entendendo a limitação:

- Considere o seguinte exemplo:

Entendendo a limitação:

- Considere o seguinte exemplo:

$$\frac{x^{15}}{15!} = \frac{x \cdot x \cdot x \cdot \cdots \cdot x \cdot x \cdot x}{1 \cdot 2 \cdot 3 \cdot \cdots \cdot 13 \cdot 14 \cdot 15}$$

Entendendo a limitação:

- Considere o seguinte exemplo:

$$\frac{x^{15}}{15!} = \frac{x \cdot x \cdot x \cdot \cdots \cdot x \cdot x \cdot x}{1 \cdot 2 \cdot 3 \cdot \cdots \cdot 13 \cdot 14 \cdot 15}$$

- Dependendo do valor de *x*, x^{15} pode ser um valor grande, mas nem sempre;

Entendendo a limitação:

- Considere o seguinte exemplo:

$$\frac{x^{15}}{15!} = \frac{x \cdot x \cdot x \cdot \cdots \cdot x \cdot x \cdot x}{1 \cdot 2 \cdot 3 \cdot \cdots \cdot 13 \cdot 14 \cdot 15}$$

- Dependendo do valor de *x*, x^{15} pode ser um valor grande, mas nem sempre;

- Além disso, *15!* é com certeza um valor grande, não sendo possível armazená-lo em um *int*.

Entendendo a limitação:

- O problema ocorre, portanto, pelo fato de ter que se calcular 1 ou 2 valores *grandes* para então se chegar a um valor *pequeno*;
  - Resultado da divisão desses valores.

Entendendo a limitação:

- O problema ocorre, portanto, pelo fato de ter que se calcular 1 ou 2 valores *grandes* para então se chegar a um valor *pequeno*;
  - Resultado da divisão desses valores.

- *Grande* nesse contexto significa um valor que não pode ser armazenado em um *int*.

Como contornar a limitação:

- Utiliza-se uma propriedade matemática para que o cálculo seja feito de maneira diferente:

Como contornar a limitação:

- Utiliza-se uma propriedade matemática para que o cálculo seja feito de maneira diferente:

$$\frac{x \cdot x \cdot \cdots \cdot x \cdot x}{1 \cdot 2 \cdot \cdots \cdot 14 \cdot 15} \xRightarrow{pode\ ser\ feito\ assim:} \frac{x}{1} \cdot \frac{x}{2} \cdot \cdots \cdot \frac{x}{14} \cdot \frac{x}{15}$$

Como contornar a limitação:

- Utiliza-se uma propriedade matemática para que o cálculo seja feito de maneira diferente:

$$\frac{x \cdot x \cdot \dots \cdot x \cdot x}{1 \cdot 2 \cdot \dots \cdot 14 \cdot 15} \xrightarrow{pode\ ser\ feito\ assim:} \frac{x}{1} \cdot \frac{x}{2} \cdot \dots \cdot \frac{x}{14} \cdot \frac{x}{15}$$

- Cada divisão é sempre realizada entre *x* e um número pequeno (1, 2, 3, ...);

Como contornar a limitação:

- Utiliza-se uma propriedade matemática para que o cálculo seja feito de maneira diferente:

$$\frac{x \cdot x \cdot \cdots \cdot x \cdot x}{1 \cdot 2 \cdot \cdots \cdot 14 \cdot 15} \xRightarrow{pode\ ser\ feito\ assim:} \frac{x}{1} \cdot \frac{x}{2} \cdot \cdots \cdot \frac{x}{14} \cdot \frac{x}{15}$$

- Cada divisão é sempre realizada entre *x* e um número pequeno (1, 2, 3, ...);

- O resultado é então acumulado em uma série de multiplicações.

Exemplo 5: Série de Taylor (e^x) (versão otimizada)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos







}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

Exemplo 5: Série de Taylor (e^x) (versão otimizada)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos
  float termo = 1;




}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

Exemplo 5: Série de Taylor (e^x) (versão otimizada)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos
  float termo = 1;
  for( j = 1 ; j <= i ; j++ ){

  }

}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

Exemplo 5: Série de Taylor (e^x) (versão otimizada)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos
  float termo = 1;
  for( j = 1 ; j <= i ; j++ ){
    termo = termo * x / j;
  }

}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

Exemplo 5: Série de Taylor (e^x) (versão otimizada)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos
  float termo = 1;
  for( j = 1 ; j <= i ; j++ ){
    termo = termo * x / j;
  }
  e_x = e_x + termo; // Acumula termo no somatório
}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

Exemplo 5: Série de Taylor (e^x) (versão otimizada)

```
float x;
int n, i, j;
printf("Digite x e a quantidade de termos n: ");
scanf("%f%d", &x, &n);
float e_x = 0; // inicializa somatório com zero
for( i = 0 ; i <= n ; i++ ){ // gera os termos
  float termo = 1;
  for( j = 1 ; j <= i ; j++ ){
    termo = termo * x / j;
  }
  e_x = e_x + termo; // Acumula termo no somatório
}
printf("e elevado a %.3f = %.8f\n", x, e_x);
```

EXERCÍCIOS

Exercícios

1. Dados N e K, mostre na tela os N primos a partir de K. Por exemplo:

   – Entrada: N = 5 e K = 10

   – Saída: 11, 13, 17, 19, 23

Exercícios

2. Faça experimentos com a versão otimizada da Série de Taylor e monte uma tabela relacionando a precisão (quantas casas decimais após a vírgula) e o valor de *n*.

   – Por exemplo, para 2 casas decimais, qual deve ser o valor de *n*?

   – Faça testes com pelo menos 2 casas decimais e vá aumentando aos poucos.

Desafio

- É possível fazer mais uma otimização no algoritmo para a Série de Taylor, reescrevendo-o com somente um laço *for*;

- O resultado esperado deve ser o mesmo do que o da 2ª versão, porém com um tempo de resposta menor;

- Analise a equação utilizada na otimização e tente descobrir como realizar o cálculo em um único laço de repetição.