

MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2015

Roteiro

- 1 Maior número
- 2 Soma de n números
- 3 Fatorial
- 4 Máximo Divisor Comum (MDC)
- 5 Números primos
- 6 Primeiros n números primos
- 7 Fatoração em números primos
- 8 Números de Fibonacci
- 9 Número de letras de uma sequência de caracteres
- 10 Conversão de números binários para decimais
- 11 Conversão de números decimais para binários
- 12 Jogo de dados
- 13 Exercícios

Introdução

- Vimos quais são os comandos de repetição em C:
 - ▶ `while`
 - ▶ `do-while`
 - ▶ `for`
- Veremos agora alguns exemplos de utilização desses comandos.

Maior número

- Vamos escrever um programa que receba n números ($n \geq 1$) e descubra qual deles é o maior.
- O programa deve ter os seguintes passos:
 - 1 Ler um número n .
 - 2 Ler n números.
 - 3 Determinar o maior entre os n números.
- Como determinar o maior?

Maior número

- A ideia é criarmos uma variável maior que armazenará o maior número lido até então.
- Algoritmo:

```
maior(n)
```

```
  ler um número maior
```

```
  repetir n-1 vezes
```

```
    ler um numero aux
```

```
    se aux > maior então
```

```
      maior = aux
```

```
  imprimir maior
```

Maior número

```
#include <stdio.h>

int main() {
    int i, n;
    double maior, aux;

    printf("Digite a quantidade de numeros: ");
    scanf("%d", &n);

    printf("Digite um numero: ");
    scanf("%lf", &maior);

    for (i = 1; i < n; i++) {
        printf("Digite um numero: ");
        scanf("%lf", &aux);
        if (aux > maior)
            maior = aux;
    }

    printf("Maior numero: %.2f\n", maior);
    return 0;
}
```

Soma de n números

- Vamos escrever um programa que receba n números quaisquer e calcule a soma deles.
- Uma variável `soma` irá armazenar a soma dos números lidos.
- Ao ler um próximo número, como atualizar a variável `soma`?
 - ▶ `soma = soma + numero;`
 - ▶ `soma += numero;`
- É importante lembrar que a variável `soma` deve ser inicializada com o valor zero (elemento neutro da soma).

Soma de n números

```
#include <stdio.h>

int main() {
    int i, n;
    double aux, soma = 0;

    printf("Quantidade de numeros: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        printf("Digite um numero: ");
        scanf("%lf", &aux);
        soma = soma + aux;
    }

    printf("Soma dos numeros: %.2f\n", soma);
    return 0;
}
```


Fatorial

- Vamos escrever um programa que leia um número inteiro positivo n e calcule o valor do seu fatorial.
- O programa deve ter os seguintes passos:
 - ① Ler um número n .
 - ② Calcular $n! = 1 \times 2 \times \dots \times (n - 1) \times n$
- Como fazer este cálculo?
- Note que n não é fixo, portanto, temos que usar comandos de repetição.

Fatorial

- A ideia é criarmos uma variável `fatorial` que na i -ésima iteração do laço de repetição possua o valor $i!$
- Note que $(i + 1)! = (i + 1) \times i!$, portanto, na $(i + 1)$ -ésima iteração podemos atualizar a variável `fatorial` da seguinte maneira:
 - ▶ `fatorial = (i+1) × fatorial`
- Algoritmo:

```
fatorial(n)
  fatorial = 1
  para i de 1 até n faça
    fatorial = i * fatorial
  imprimir fatorial
```

- Note que a variável `fatorial` deve ser inicializada com o valor um (elemento neutro da multiplicação).

Fatorial

```
#include <stdio.h>

int main() {
    int i, n;
    unsigned int fatorial = 1;

    printf("Digite numero inteiro (nao negativo): ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++)
        fatorial = fatorial * i;

    printf("%d! = %d\n", n, fatorial);

    return 0;
}
```

Fatorial

- No exemplo anterior, o fatorial é calculado corretamente para $n \leq 14$, entretanto, falha para $n \geq 15$.
- Por quê? Como resolver este problema?
 - ▶ Podemos trocar o tipo da variável fatorial de `unsigned int` para `unsigned long int` ou mesmo por `double`.
 - ▶ Com `unsigned long int` é possível calcular corretamente até o fatorial de 20, enquanto que, com `double`, até 170 (neste caso, com perda de precisão numérica).

Fatorial

```
#include <stdio.h>

int main() {
    unsigned short int i, n;
    unsigned long int fatorial = 1;

    printf("Digite numero inteiro (nao negativo): ");
    scanf("%hu", &n);

    for (i = 1; i <= n; i++)
        fatorial = fatorial * i;

    printf("%hu! = %lu\n", n, fatorial);

    return 0;
}
```

Fatorial

```
#include <stdio.h>

int main() {
    unsigned short int i, n;
    double fatorial = 1;

    printf("Digite numero inteiro (nao negativo): ");
    scanf("%hu", &n);

    for (i = 1; i <= n; i++)
        fatorial = fatorial * i;

    printf("%hu! = %.0f\n", n, fatorial);

    return 0;
}
```

Máximo Divisor Comum (MDC)

- O algoritmo de Euclides para o cálculo do Máximo Divisor Comum (MDC) entre dois números inteiros positivos m e n , apresentado em 300 a.C., é um dos algoritmos mais antigos do mundo.
- O algoritmo pode ser resumido na seguinte fórmula:

$$\text{mdc}(m, n) = \begin{cases} m, & \text{se } n = 0 \\ \text{mdc}(n, m \% n), & \text{se } n > 0 \end{cases}$$

Máximo Divisor Comum (MDC)

```
#include <stdio.h>

int main() {
    int m, n, aux;

    printf("Entre com dois numeros inteiros positivos: ");
    scanf("%d %d", &m, &n);

    while (n > 0) {
        aux = n;
        n = m % n;
        m = aux;
    }

    printf("MDC = %d\n", m);

    return 0;
}
```


Números primos

- Um número é primo se ele tem exatamente dois divisores naturais distintos: o número um e ele mesmo.
- O programa deve ter os seguintes passos:
 - 1 Ler um número n .
 - 2 Testar se nenhum dos números entre 2 e $n - 1$ divide n .
- Lembre-se que o operador $\%$ retorna o resto da divisão inteira.
- Portanto $(a \% b)$ é zero se, e somente se, b divide a .
- Note que não é necessário testar os números entre $\lfloor n/2 \rfloor + 1$ e $n - 1$.
- De fato é possível testar menos números ainda...
 - 1 Se $n = a \times b$, sendo n , a e b números naturais e $a \leq b$, então $a \leq \sqrt{n} \leq b$.
 - 2 Logo, precisamos testar apenas os números inteiros entre 2 e $\lfloor \sqrt{n} \rfloor$.

Números primos

```
primo(n)
  aux = 2
  primo = verdadeiro
  enquanto primo e aux <= n/2 faça
    se aux for um divisor de n então
      primo = falso
    aux = aux + 1
  imprimir primo
```

Números primos

```
#include <stdio.h>

int main() {
    unsigned int aux, n, primo = 1;

    printf("Digite um numero inteiro: ");
    scanf("%u", &n);

    for (aux = 2; primo && (aux <= n/2); aux++)
        if ((n % aux) == 0)
            primo = 0;

    if (primo)
        printf("Numero primo\n");
    else
        printf("Numero composto\n");

    return 0;
}
```

Números primos

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

int main() {
    unsigned int aux, n, primo = 1;

    printf("Digite um numero inteiro: ");
    scanf("%u", &n);

    for (aux = 2; primo && (aux <= sqrt(n)); aux++)
        if ((n % aux) == 0)
            primo = 0;

    if (primo)
        printf("Numero primo\n");
    else
        printf("Numero composto\n");

    return 0;
}
```

Primeiros n números primos

- Já sabemos testar se um dado número é primo.
- Como fazer para imprimir os n primeiros números primos?

Primeiros n números primos

```
#include <stdio.h>

int main() {
    int aux, numero = 2, primos = 0, n, primo;

    printf("Digite o numero de primos a serem calculados: ");
    scanf("%d",&n);

    while (primos < n) {

        /* inserir aqui trecho de codigo que
           verifica se o numero eh primo */

        if (primo) {
            printf("%d\n", numero);
            primos++;
        }

        numero++;
    }

    return 0;
}
```

Primeiros n números primos

```
while (primos < n) {  
  
    /* inserir aqui trecho de código que  
       verifica se o número é primo */  
  
    if (primo) {  
        printf("%d\n", numero);  
        primos++;  
    }  
  
    numero++;  
}
```

Primeiros n números primos

```
while (primos < n) {
    primo = 1;

    for (aux = 2; primo && (aux <= numero/2); aux++)
        if (numero % aux == 0)
            primo = 0;

    if (primo) {
        printf("%d\n", numero);
        primos++;
    }

    numero++;
}
```


Primeiros n números primos

- Note que o número 2 é o único número par que é primo.
- Podemos alterar o programa para primeiro imprimir o número 2...

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

```
int main() {
```

```
    int aux, numero, primos = 1, n, primo;
```

```
    printf("Digite o numero de primos a serem calculados: ");
```

```
    scanf("%d",&n);
```

```
    if (n > 0) {
```

```
        printf("%d\n", 2);
```

```
        ...
```

Primeiros n números primos

- ... e depois imprimir os primos ímpares:

...

```
for (numero = 3; primos < n; numero = numero + 2) {  
    primo = 1;
```

```
    for (aux = 2; primo && (aux <= numero/2); aux++)  
        if (numero % aux == 0)  
            primo = 0;
```

```
    if (primo) {  
        printf("%d\n", numero);  
        primos++;  
    }
```

```
}
```

```
}
```

```
return 0;
```

```
}
```

Primeiros n números primos

- ... e depois imprimir os primos ímpares:

...

```
for (numero = 3; primos < n; numero = numero + 2) {
    primo = 1;

    for (aux = 3; primo && (aux <= numero/2); aux = aux + 2)
        if (numero % aux == 0)
            primo = 0;

    if (primo) {
        printf("%d\n", numero);
        primos++;
    }
}

return 0;
}
```

Fatoração em números primos

- Dado um número inteiro positivo n , como fatorá-lo em números primos?
- Podemos verificar todos os candidatos a fatores de n , ou seja, números menores ou iguais a n .
- Seja *fator* o número que está sendo testado numa certa iteração. Se *fator* for um divisor de n , então imprima *fator* e divida n por *fator*. Caso contrário, incremente *fator*.
- Note que não é necessário testar se *fator* é um número primo.

Fatoração em números primos

```
#include <stdio.h>

int main() {
    unsigned int n, fator = 2;

    printf("Digite um numero inteiro positivo: ");
    scanf("%u", &n);
    printf("%d = 1", n);

    while (n > 1) {
        if ((n % fator) == 0) {
            printf(" x %d", fator);
            n = n / fator;
        } else
            fator++;
    }

    printf("\n");
    return 0;
}
```

Números de Fibonacci

- A série de Fibonacci é a seguinte:
 - ▶ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...
- Ou seja, o n -ésimo termo da série é dado pela seguinte fórmula:

$$F(n) = \begin{cases} 1, & \text{se } n = 1 \\ 1, & \text{se } n = 2 \\ F(n-1) + F(n-2), & \text{caso contrário} \end{cases}$$

- Como escrever um algoritmo para imprimir os primeiros n números da série?

Números de Fibonacci

```
fibonacci(n)
  atual = 1
  proximo = 1
  para i de 1 até n faça
    imprimir atual
    temp = atual + proximo
    atual = proximo
    proximo = temp
```

Números de Fibonacci

```
#include <stdio.h>

int main() {
    unsigned long int n, atual = 1, proximo = 1, temp, i;

    printf("Digite um numero inteiro positivo: ");
    scanf("%lu", &n);

    for (i = 1; i <= n; i++) {
        printf("%lu\n", atual);
        temp = atual + proximo;
        atual = proximo;
        proximo = temp;
    }

    return 0;
}
```


Número de letras de uma sequência de caracteres

- Considere o seguinte problema: dada uma sequência de caracteres em uma única linha (uma frase, por exemplo), determinar o seu número de letras.
- Exemplo:
 - ▶ Entrada: Um4 seQu3nc!4de c4r4ct3r3S
 - ▶ Saída: 16 letras
- Como determinar se um caractere é uma letra?
 - ▶ Letras maiúsculas: de 65 ('A') a 90 ('Z') na tabela ASCII
 - ▶ Letras minúsculas: de 97 ('a') a 122 ('z') na tabela ASCII
- Como determinar o fim da sequência de caracteres?

Número de letras de uma sequência de caracteres

```
#include <stdio.h>

int main() {
    char c;
    int letras = 0;

    printf("Digite uma frase: ");

    do {
        scanf("%c", &c);

        if (((c >= 65) && (c <= 90)) || ((c >= 97) && (c <= 122)))
            letras++;
    } while (c != '\n');

    printf("%d letras\n", letras);

    return 0;
}
```

Número de letras de uma sequência de caracteres

```
#include <stdio.h>

int main() {
    char c;
    int letras = 0;

    printf("Digite uma frase: ");

    do {
        scanf("%c", &c);

        if (((c >= 'A') && (c <= 'Z')) || ((c >= 'a') && (c <= 'z')))
            letras++;
    } while (c != '\n');

    printf("%d letras\n", letras);

    return 0;
}
```

Conversão de números binários para decimais

- *“There are 10 types of people in the world: those who understand binary and those who don’t.” – Unknown*
- Sabemos que um computador armazena todas as informações na forma binária, portanto, é útil saber como converter números decimais em binários (e vice-versa).
- Dado um número binário $b_n b_{n-1} \dots b_2 b_1 b_0$, este corresponde na forma decimal a:

$$\sum_{i=0}^n b_i \times 2^i$$

- Exemplos:

$$101 = 2^2 + 2^0 = 5$$

$$1001110100 = 2^9 + 2^6 + 2^5 + 2^4 + 2^2 = 512 + 64 + 32 + 16 + 4 = 628$$

Conversão de números binários para decimais

- Vamos supor que lemos um inteiro binário.
- Ou seja, ao lermos $n = 111$ assumimos que este é um número binário (e não “cento e onze”).
- Como transformar este número no correspondente valor decimal (7, neste caso)?
- Podemos usar a expressão:

$$\sum_{i=0}^n b_i \times 2^i$$

Para isso, entretanto, devemos conseguir recuperar os dígitos binários (bits) individualmente.

- Note que:
 - ▶ $n \% 10$ recupera o último dígito de n .
 - ▶ $n = n / 10$ remove o último dígito de n .

Conversão de números binários para decimais

```
#include <stdio.h>

int main() {
    unsigned long int n, dec = 0, pot = 1, bit;

    printf("Digite um numero binario: ");
    scanf("%lu", &n);

    while (n > 0) {
        bit = n % 10;
        n = n / 10;
        dec = dec + (bit * pot);
        pot = pot * 2;
    }

    printf("%lu\n", dec);
    return 0;
}
```

Conversão de números decimais para binários

- Agora, dado um número em decimal, como obter o correspondente em binário?
- Qualquer número pode ser escrito como uma soma de potências de 2:

$$6 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$13 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

- O que acontece se dividirmos, sucessivamente, um número decimal por 2?
- Vamos testar com o número 13:

$$13/2 = 6, \text{ com resto } 1$$

$$6/2 = 3, \text{ com resto } 0$$

$$3/2 = 1, \text{ com resto } 1$$

$$1/2 = 0, \text{ com resto } 1$$

Conversão de números decimais para binários

```
#include <stdio.h>

int main() {
    unsigned long int n, bin = 0, pot = 1, bit;

    printf("Digite um numero decimal: ");
    scanf("%lu", &n);

    while (n > 0) {
        bit = n % 2;
        n = n / 2;
        bin = bin + (bit * pot);
        pot = pot * 10;
    }

    printf("%lu\n", bin);
    return 0;
}
```


Jogo de dados

- Suponha que queremos imprimir todas as possibilidades de resultados ao se jogar 4 dados de 6 faces.
- Para cada possibilidade do primeiro dado, devemos imprimir todas as possibilidades dos 3 dados restantes.
- Para cada possibilidade do primeiro e do segundo dado, devemos imprimir todas as possibilidades dos 2 dados restantes e assim por diante.

Jogo de dados

```
#include <stdio.h>

int main() {
    int d1, d2, d3, d4;

    printf("D1 D2 D3 D4\n");

    for (d1 = 1; d1 <= 6; d1++)
        for (d2 = 1; d2 <= 6; d2++)
            for (d3 = 1; d3 <= 6; d3++)
                for (d4 = 1; d4 <= 6; d4++)
                    printf("%d %d %d %d\n", d1, d2, d3, d4);

    return 0;
}
```

Exercícios

- Um número inteiro é dito perfeito se a soma dos seus divisores positivos (excluindo ele mesmo) é igual ao próprio número. Dado um número inteiro n , escreva um programa que determine se ele é perfeito.
- Dado um número x qualquer e um número inteiro não negativo n , escreva um programa para calcular o valor de x^n , sem usar a biblioteca matemática (`math.h`).
- Dados dois números x e y ($x \geq 1$, $y > 1$), escreva um programa para calcular o valor de $\lfloor \log_y x \rfloor$, sem usar a biblioteca matemática (`math.h`).
- Escreva um programa que, dados dois números inteiros positivos, calcule o Mínimo Múltiplo Comum (MMC) deles.
- Dado um número inteiro positivo n escrito na base x , converta-o para a base y , sendo que x e y também são fornecidos como entrada do seu programa ($2 \leq x, y \leq 10$).