Programação Estruturada

Estruturas de repetição

Professores Emílio Francesquini e Carla Negri Lintzmayer

2018.Q3

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Comandos de repetição

Comandos de repetição

- Até agora vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.
- Entretanto, eventualmente é necessário executar um bloco de comandos várias vezes para se obter o resultado esperado.

Repetição

Vamos imprimir todos os números de 1 até 4.

```
1 printf("1\n");
2 printf("2\n");
3 printf("3\n");
4 printf("4\n");
```

Dá para fazer com o que já vimos.

Repetição

Vamos imprimir todos os números de 1 até 100.

```
1  printf("1\n");
2  printf("2\n");
3  printf("3\n");
4  printf("4\n");
5  printf("5\n");
6  ...
7  printf("100\n");
```

Ainda dá para fazer com o que já vimos, mas é mais chato.

Repetição

Vamos imprimir todos os números de 1 até n, onde n é informado pelo usuário.

```
1  scanf("%d", &n);
2  if (n >= 1)
3      printf("1\n");
4  if (n >= 2)
5      printf("2\n");
6  if (n >= 3)
7      printf("3\n");
8  ...
9  if (n >= 100)
10     printf("100\n");
```

Agora ficou impossível!

Note que esse programa é válido apenas para $n \leq 100$.

Comando while

Comando while

Estrutura:

```
1 while (condição)
2     comando;
```

ou

```
1 while (condição) {
2     comandos;
3 }
```

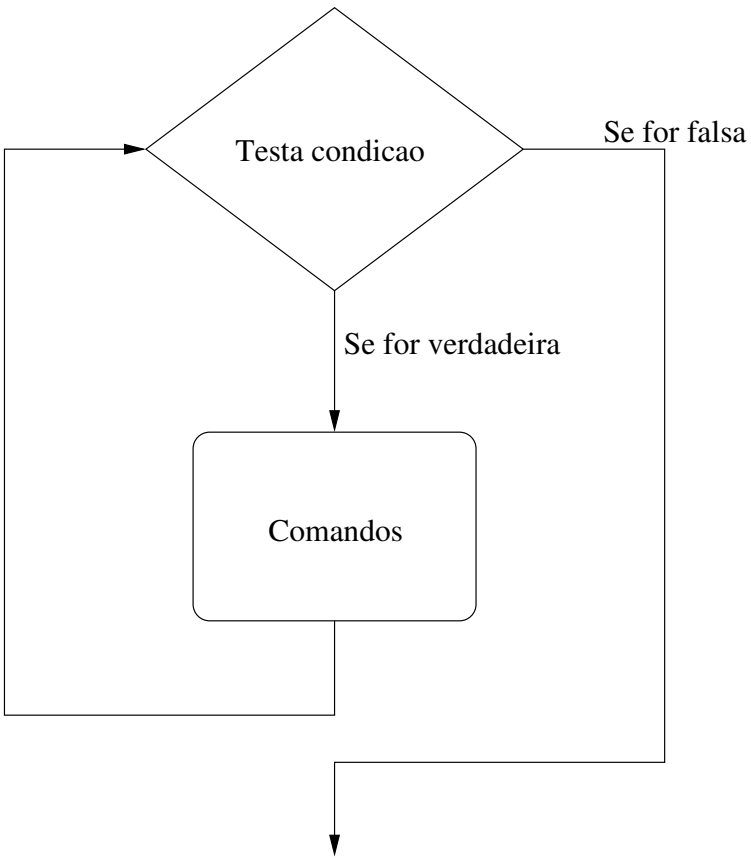
Enquanto a condição for verdadeira ($\neq 0$), o(s) comando(s) são executados.

Comando while

Passo 1 Testa a condição. Se ela for verdadeira, vai para o Passo 2. Se for falsa, vai para o Passo 3.

Passo 2 Executa os comandos. Vai para o Passo 1.

Passo 3 Segue o programa.



Comando while

Vamos imprimir todos os números de 1 até 100.

```c
#include <stdio.h>

int main() {
    int i;

    i = 1;
    while (i <= 100) {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Comando while

Vamos imprimir todos os números de 1 até n.

```c
#include <stdio.h>

int main() {
    int i, n;

    scanf("%d", &n);
    i = 1;
    while (i <= n) {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

Comando while

- O que acontece se a condição for falsa na primeira vez?

```
1  while (a != a)
2      a = a + 1;
```

- O que acontece se a condição for sempre verdadeira?

```
1  while (a == a)
2      a = a + 1;
```

Comando do-while

Comando do-while

Estrutura:

```
1 do
2     comando;
3 while (condição);
```

ou

```
1 do {
2     comandos;
3 } while (condição);
```

Diferença do **while**: sempre executa o(s) comando(s) pelo menos uma vez.

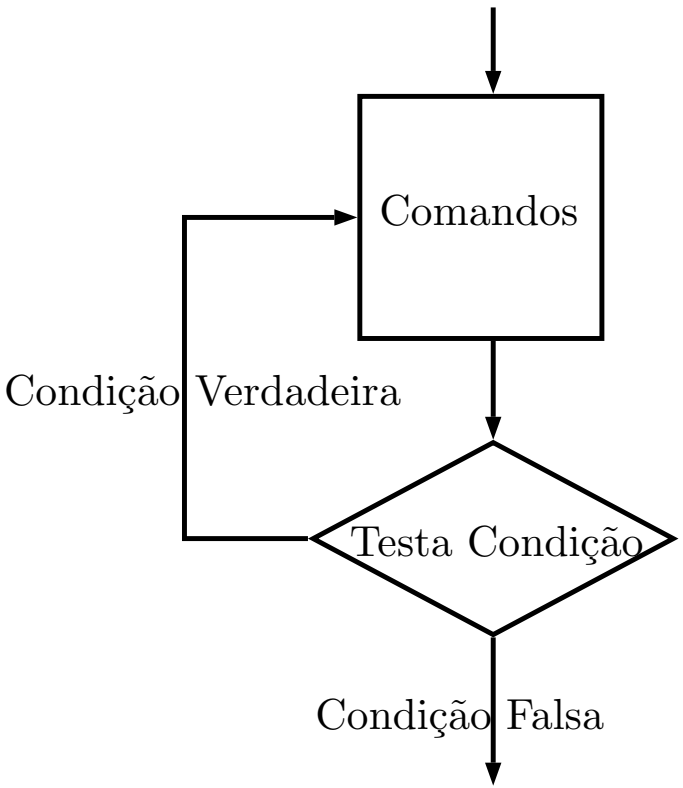
O teste condicional é feito por último.

Comando do-while

Passo 1 Executa os comandos. Vai para o Passo 2.

Passo 2 Testa a condição. Se ela for verdadeira, vai para Passo 1. Se for falsa, vai para o Passo 3.

Passo 3 Segue o programa.



Comando do-while

Vamos imprimir todos os números de 1 até 100.

```c
#include <stdio.h>

int main() {
    int i;

    i = 1;
    do {
        printf("%d\n", i);
        i = i + 1;
    } while (i <= 100);

    return 0;
}
```

Comando do-while

Vamos imprimir todos os números de 1 até n.

```c
#include <stdio.h>

int main() {
    int i, n;

    scanf("%d", &n);
    i = 1;
    do {
        printf("%d\n", i);
        i = i + 1;
    } while (i <= n);

    return 0;
}
```

E se o usuário digitar 0?

O comando for

O comando for

Estrutura:

```
1 for (início; condição; passo)
2     comando;
```

ou

```
1 for (início; condição; passo) {
2     comandos;
3 }
```

- **Início**: uma ou mais atribuições, separadas por “,”
- **Condição**: os comandos são executados enquanto a condição for verdadeira
- **Passo**: um ou mais comandos separados por “,”. Os comandos do passo sempre são executados após os comandos do bloco

O comando for

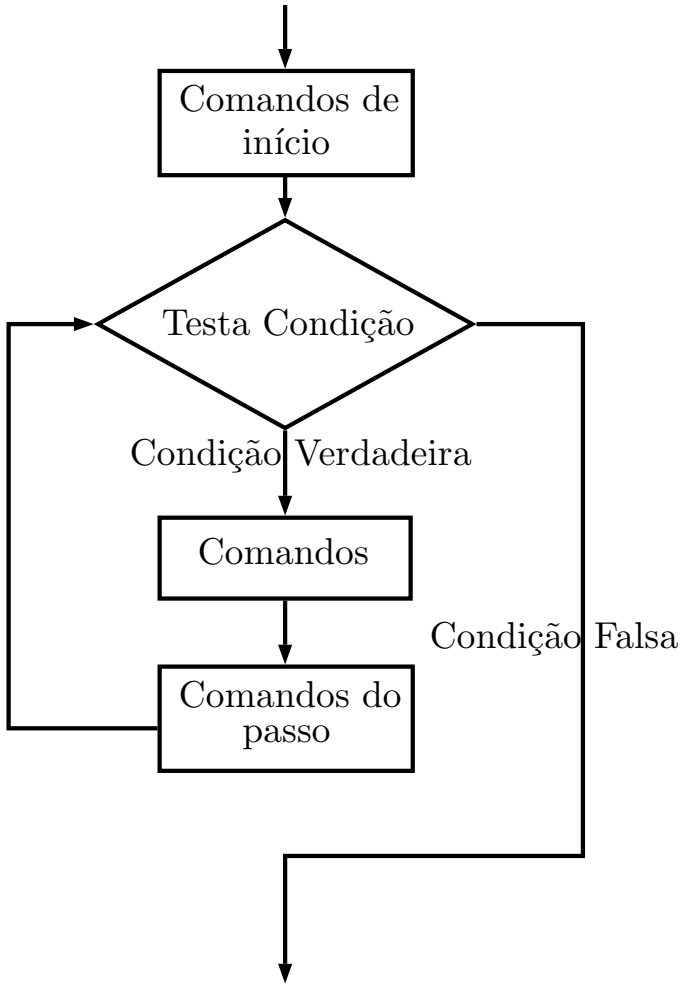
Passo 1 Executa os comandos em "início". Vai para Passo 2.

Passo 2 Testa a condição. Se ela for verdadeira, vai para Passo 3. Se for falsa, vai para Passo 5.

Passo 3 Executa os comandos do bloco. Vai para Passo 4.

Passo 4 Executa os comandos em "passo". Vai para Passo 2.

Passo 5 Segue o programa.



O comando for

O **for** é equivalente à seguinte construção utilizando o **while**:

```
1 início;
2 while (condição) {
3     comandos;
4     passo;
5 }
```

O comando for

Vamos imprimir todos os números de 1 até 100.

```c
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i <= 100; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```

O comando for

Vamos imprimir todos os números de 1 até n.

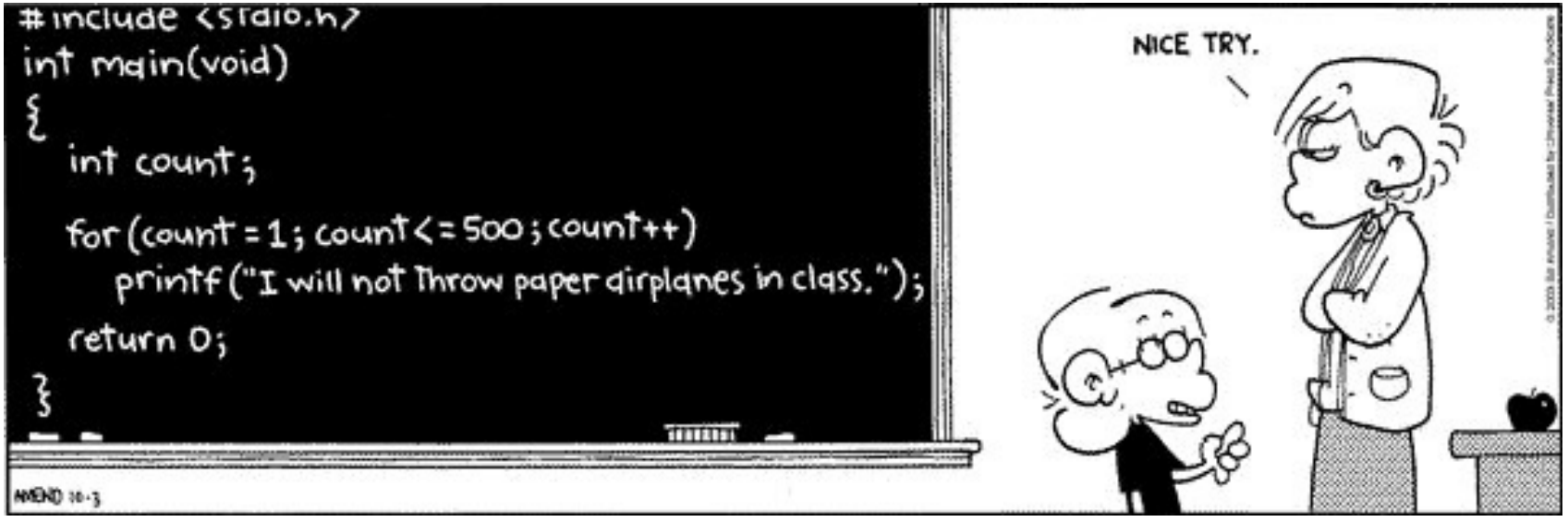
```c
#include <stdio.h>

int main() {
    int i, n;

    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```

I'll not throw paper airplanes in class



Comandos continue e break

Laços e o comando break

O comando **break** faz com que a execução de um laço seja terminada, passando a execução para o próximo comando depois do final do laço.

```
1  int i;
2
3  for (i = 1; i <= 10; i++) {
4      if (i >= 5)
5          break;
6      printf("%d\n",i);
7  }
8
9  printf("Terminou o laço\n");
```

O que será impresso?

Laços e o comando break

Assim como a "condição" em laços, o comando **break** é utilizado em situações de parada de um laço.

Ex.: Imprimindo os números de 1 até 10.

```
1  int i;
2  for (i = 1; ; i++) {
3      if (i > 10)
4          break;
5      printf("%d\n", i);
6  }
```

é equivalente a:

```
1  int i;
2  for (i = 1; i <= 10; i++) {
3      printf("%d\n", i);
4  }
```

Laços e o comando continue

O comando **continue** faz com que a execução de um laço seja alterada para o final do laço.

```
1  int i;
2  for (i = 1; i <= 10; i++) {
3      if (i == 5)
4          continue;
5      printf("%d\n", i);
6  }
7  printf("Terminou o laço\n");
```

O que será impresso?

Laços e o comando continue

O **continue** é utilizado em situações onde comandos dentro do laço só devem ser executados caso alguma condição seja satisfeita.

Ex.: Imprimindo área de um círculo, mas apenas se o raio for par e estiver entre 1 e 10.

```
1  int r;
2  double area;
3  for (r = 1; r <= 10; r++) {
4      if ((r % 2) != 0) /* se número for ímpar pulamos */
5          continue;
6      area = 3.1415 * r * r;
7      printf("%lf\n", area);
8  }
```

Mas note que poderíamos escrever algo mais simples:

Laços encaixados

Laços encaixados

- Para resolver alguns problemas, é necessário implementar um laço dentro de outro laço.
- Estes são conhecidos como laços encaixados (ou aninhados).

```c
int main() {
    int i, j;

    for (i = 1; i <= 4; i++)
        for (j = 1; j <= 3; j++)
            printf("%d %d\n", i, j);

    return 0;
}
```

- O que será impresso por este programa?

Laços encaixados

```
1  for (i = 1; i <= 4; i++)
2      for (j = 1; j <= 3; j++)
3          printf("%d %d\n", i, j);
```

- Fixado um valor para **i** no primeiro laço **for**, começa-se o segundo laço **for**, que varia o valor de **j** entre 1 e 3.
- No final deste segundo laço **for** voltamos para o primeiro laço, onde a variável **i** assumirá seu próximo valor.
- Fixado este valor de **i** começa-se novamente o segundo laço **for**.

Laços encaixados

```
1  for (i = 1; i <= 4; i++)
2      for (j = 1; j <= 3; j++)
3          printf("%d %d\n", i, j);
```

Será impresso:

```
1  1 1
2  1 2
3  1 3
4  2 1
5  2 2
6  ...
7  4 1
8  4 2
9  4 3
```

Exemplos com laços

Variável acumuladora

- Vamos ver alguns exemplos de problemas que são resolvidos utilizando laços.
- Há alguns padrões de solução que são bem conhecidos e são úteis em diversas situações.
- O primeiro padrão deles é o uso de uma "variável acumuladora".

Soma de números

Problema

Ler um inteiro positivo n, em seguida ler n números do teclado e apresentar a soma destes.

Soma de números

- Como n não é definido a priori, não podemos criar n variáveis e depois somar seus valores.
- A ideia é criar uma variável acumuladora que a cada iteração de um laço guarda a soma de todos os números lidos *até então*.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração ela tem a soma dos $(i-1)$ números lidos anteriormente.
  - Durante a i-ésima iteração ela soma a seu valor o novo número lido.

```
1  acumuladora = 0 /* no início ainda não somamos nada */
2  repita n vezes:
3      leia um número aux
4      acumuladora = acumuladora + aux
```

Soma de números

- Podemos usar qualquer estrutura de laço de C para esta solução.
- Abaixo temos uma solução utilizando o comando **for**.

```
1  printf("Digite o valor de n: ");
2  scanf("%d", &n);
3
4  soma = 0;
5  for (i = 1; i <= n; i++) {
6      printf("Digite um novo número: ");
7      scanf("%d", &aux);
8      soma = soma + aux;
9  }
```

Soma de números: código completo

```c
#include <stdio.h>

int main() {
    int i, n, soma, aux;

    printf("Digite o valor de n: ");
    scanf("%d", &n);

    soma = 0;
    for (i = 1; i <= n; i++) {
        printf("Digite um numero: ");
        scanf("%d", &aux);
        soma = soma + aux;
    }

    printf("Soma: %d\n", soma);

    return 0;
}
```

Calculando a divisão

Problema

Calcular a divisão inteira de dois numeros usando apenas soma e subtração.

Algoritmo solução

```
1  leia dividendo e divisor
2  contador = 0
3  enquanto dividendo >= divisor
4      dividendo = dividendo - divisor
5      contador = contador + 1
6  exiba contador
7  /* note que dividendo contém o resto da divisão */
```

Por que?

Contador equivale à divisão inteira de dividendo por divisor.

Divisão inteira

```c
int main() {
    int dividendo, divisor, contador, aux;
    printf("Entre com o dividendo: ");
    scanf("%d", &dividendo);
    printf("Entre com o divisor: ");
    scanf("%d", &divisor);

    contador = 0;
    aux = dividendo;
    while (aux >= divisor) {
        aux = aux - divisor;
        contador++;
    }

    printf("A divisao de %d por %d eh %d e tem resto igual a %d.\n",
    ↪  dividendo, divisor, contador, aux);

    return 0;
}
```

Variável indicadora

- Um outro uso comum de laços é para verificar se um determinado objeto, ou conjunto de objetos, satisfaz uma propriedade ou não.
- Um padrão que pode ser útil na resolução deste tipo de problema é o uso de uma **variável indicadora**.
    - Assumimos que o objeto satisfaz a propriedade (indicadora = Verdade).
    - Com um laço verificamos se o objeto realmente satisfaz a propriedade.
    - Se em alguma iteração descobrirmos que o objeto não satisfaz a propriedade, então fazemos indicadora = Falso.

Números primos

A geração de números primos é uma parte fundamental em sistemas criptográficos como os utilizados em *internetbanking*.

Problema

Determinar se um número n é primo ou não.

Números primos

- Um número é primo se seus únicos divisores são 1 e ele mesmo.
- Dado um número n, como detectar se este é ou não primo??
  - Podemos testar se nenhum dos números entre 2 e $(n-1)$ divide n.
- Lembre-se que o operador % retorna o resto da divisão.
- Portanto $(n\%b)$ é zero se e somente se b divide n.

Números primos

```
1  leia um número e salve em n
2  div = 2
3  indicadora = 1 /* assumimos que n é primo */
4  enquanto div <= (n-1) faça
5      se (n % div) == 0 então
6          indicadora = 0 /* descobrimos que n não é primo */
7      div = div + 1
8  se indicadora == 1 então o número é primo
```

Números primos

```c
int main() {
    int div, n, eh_primo;
    printf("Digite um número:");
    scanf("%d", &n);

    div = 2;
    eh_primo = 1;
    while (div <= n-1) {
        if (n % div == 0)
            eh_primo = 0;
        div++;
    }

    if (eh_primo)
        printf("É primo!\n");
    else
        printf("Não é primo!\n");

    return 0;
}
```

Números primos

Note que assim que descobrirmos que n não é primo, podemos parar o laço.

```c
int main() {
    int div, n, eh_primo;

    printf("Digite um número:");
    scanf("%d", &n);

    div = 2;
    eh_primo = 1;
    while (div <= n-1 && eh_primo) { /* se eh_primo == 0 podemos sair do laço */
        if (n % div == 0)
            eh_primo = 0;
        div++;
    }

    if (eh_primo)
        printf("É primo!\n");
    else
        printf("Não é primo!\n");

    return 0;
}
```

Números primos

Podemos parar o laço com o uso de **break**.

```c
int main() {
    int div, n, eh_primo;

    printf("\n Digite um número:");
    scanf("%d", &n);

    div = 2;
    eh_primo = 1;
    while (div <= n-1) {
        if (n % div == 0) {
            eh_primo = 0;
            break;
        }
        div++;
    }

    if (eh_primo)
        printf("É primo!\n");
    else
        printf("Não é primo!\n");

    return 0;
}
```

Variável contadora

- Considere ainda o uso de laços para verificar se um determinado objeto, ou conjunto de objetos, satisfaz uma propriedade ou não.
- Um outro padrão que pode ser útil é o uso de uma **variável contadora**.
  - Esperamos que um objeto satisfaça x vezes uma sub-propriedade.
  - Usamos um laço e uma variável que **conta** o número de vezes que o objeto tem a sub-propriedade satisfeita.
  - Ao terminar o laço, se contadora for igual à x então o objeto satisfaz a propriedade.

Variável contadora: primeiros primos

Problema

Imprimir os n primeiros números primos.

Variável contadora: primeiros primos

O programa abaixo verifica se o valor na variável **candidato** corresponde a um primo:

```
1  divisor = 2;
2  eh_primo = 1;
3  while (divisor <= candidato/2 && eh_primo) {
4      if (candidato % divisor == 0)
5          eh_primo = 0;
6      divisor++;
7  }
8
9  if (eh_primo) {
10     printf("%d, ", candidato);
11 }
```

Variável contadora: primeiros primos

Criamos um laço externo e usamos uma variável contadora **primosImpressos**, que contará o número de primos impressos durante a execução deste laço.

```
1  while (primosImpressos < n) {
2      /* trecho do código anterior que checa se candidato é
       ↪  ou não é primo */
3
4      if (eh_primo) {
5          printf("%d, ", candidato);
6          primosImpressos++;
7      }
8
9      candidato++; /* testa próximo candidato a primo */
10 }
```

Variável contadora: primeiros primos

- Incluímos uma parte inicial de código para leitura de **n** e inicialização de variáveis.
- Para finalizar, basta incluir o trecho de código que checa se um número é primo ou não.

Variável contadora: primeiros primos

```c
int main() {
    int divisor, candidato, primosImpressos, n, eh_primo;

    printf("Digite um número inteiro positivo: ");
    scanf("%d",&n);

    candidato = 2;
    primosImpressos = 0;
    while (primosImpressos < n) {
        divisor = 2;
        eh_primo = 1;
        while (divisor <= candidato/2 && eh_primo) {
            if (candidato % divisor == 0)
                eh_primo = 0;
            divisor++;
        }

        if (eh_primo) {
            printf("%d, ", candidato);
            primosImpressos++;
        }

        candidato++; /* testa próximo número candidato a primo */
    }

    return 0;
}
```

Variável contadora: primeiros primos

O que acontece se mudarmos a variável indicadora **eh_primo** para fora do primeiro laço **while**? Faz diferença?

```c
int main() {
    int divisor, candidato, primosImpressos, n, eh_primo;
    printf("Digite um número inteiro positivo: ");
    scanf("%d", &n);
    candidato = 2;
    primosImpressos = 0;
    eh_primo = 1;  /* fora do laço, faz diferença? */
    while (primosImpressos < n) {
        divisor = 2;
        while (divisor <= candidato/2 && eh_primo) {
            if (candidato % divisor == 0)
                eh_primo = 0;
            divisor++;
        }
        if (eh_primo) {
            printf("%d, ", candidato);
            primosImpressos++;
        }
        candidato++; /* testa próximo número candidato a primo */
    }
    return 0;
}
```

Variável contadora: primeiros primos

- O que acontece se mudarmos a variável indicadora **eh_primo** para fora do primeiro laço **while**? Faz diferença?
- Resposta: Quando testarmos um **candidato** que não é primo, a variável **eh_primo** será setada para **0** e nunca mais será setada para **1**.
- Logo, nenhum outro **candidato** posterior será identificado como primo.

Variável contadora: primeiros primos

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

```
1  int main() {
2      int divisor, candidato, primosImpressos, n, eh_primo;
3
4      printf("\n Digite um número inteiro positivo:");
5      scanf("%d",&n);
6
7      if (n > 0) {
8          printf("%d, ", 2);
9          .....
```

Variável contadora: primeiros primos

Podemos alterar o programa para testar apenas números ímpares como candidatos a primo:

```
1       .....
2       candidato = 3;
3       primosImpressos = 1;
4       while (primosImpressos < n) {
5           divisor = 2;
6           eh_primo = 1;
7           while (divisor <= candidato/2 && eh_primo) {
8               if (candidato % divisor == 0)
9                   eh_primo = 0;
10              divisor++;
11          }
12
13          if (eh_primo) {
14              printf("%d, ", candidato);
15              primosImpressos++;
16          }
17
18          candidato += 2; /* testa próximo número (ímpar) candidato a primo */
19      }
20
21      return 0;
22  }
```

Variável contadora: primeiros primos

Além disso, sabendo que **candidato** é sempre um número ímpar:

- Não precisamos mais testar os divisores que são pares.
- Se **candidato** é sempre um número ímpar, ele não pode ser divisível por um número par, pois se não seria divisível por 2 também.
- Portanto basta testar divisores ímpares.

Variável contadora: primeiros primos

```c
int main() {
    int divisor, candidato, primosImpressos, n, eh_primo;

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

    if (n > 0) {
        printf("%d, ", 2);
        candidato = 3;
        primosImpressos = 1;
        while (primosImpressos < n) {
            divisor = 3; /* primeiro divisor ímpar a ser testado */
            eh_primo = 1;
            while (divisor <= candidato/2 && eh_primo) {
                if (candidato % divisor == 0)
                    eh_primo = 0;
                divisor = divisor + 2; /* demais possíveis divisores são ímpares */
            }

            if (eh_primo) {
                printf("%d, ", candidato);
                primosImpressos++;
            }
            candidato = candidato + 2; /* testa próximo número candidato a primo */
        }
    }
    return 0;
}
```

Outros exemplos

Outros exemplos

- O uso de variáveis **acumuladoras**, **indicadoras** e **contadoras** são úteis em várias situações.
- Mas não existem fórmulas para a criação de soluções para problemas.
- Em outros problemas, o uso destes padrões pode aparecer em conjunto, ou nem mesmo aparecer como parte da solução.

Maior número

Problema

Fazer um programa que lê n números do teclado e informa qual foi o maior número lido.

Maior número

- O programa deve ter os seguintes passos:
  1. Leia um número e salve em n.
  2. Repita n vezes a leitura de um número determinando o maior.
- Mas como determinar o maior?

Maior número

- A ideia é criar uma variável **maior** que sempre armazena o maior número lido até então.

```
1  leia um número e salve em n
2  leia um número e salve em maior
3  repita n-1 vezes:
4      leia um número e salve em aux
5      se aux > maior então
6          maior = aux
```

Maior número

```c
int main() {
    int cont, n, maior, aux;

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

    printf("Digite um número: ");
    scanf("%d", &maior); /* com um número lido, ele é o maior */
    cont = 1; /* já lemos um número */
    while (cont < n) {
        printf("Digite um número: ");
        scanf("%d", &aux);
        if (aux > maior)
            maior = aux;
        cont++;
    }
    printf("O maior numero lido é: %d\n", maior);

    return 0;
}
```

Equações lineares inteiras

Um uso comum de laços encaixados ocorre quando para cada um dos valores de uma determinada variável, precisamos gerar/checar algo sobre os valores de outras variáveis.

Problema

Determinar todas as soluções inteiras de um sistema linear como

$$x_1 + x_2 = C$$

com $x_1 \geq 0$, $x_2 \geq 0$, $C \geq 0$ e todos valores inteiros.

Equações lineares inteiras

Uma solução possível: para cada um dos valores de x_1, com $0 \leq x_1 \leq C$, teste todos os valores de x_2 possíveis e verifique quais deles são soluções.

```
1  para cada x1 entre 0 e C faça:
2      para cada x2 entre 0 e C faça:
3          se x1 + x2 = C então imprima solução
```

Equações lineares inteiras

```c
int main() {
    int C, x1, x2;

    printf("Digite o valor de C: ");
    scanf("%d", &C);

    for (x1 = 0; x1 <= C; x1++) {
        for (x2 = 0; x2 <= C; x2++) {
            if (x1 + x2 == C)
                printf("%d + %d = %d\n", x1, x2, C);
        }
    }

    return 0;
}
```

Equações lineares inteiras

Note que, fixado x_1, não precisamos testar todos os valores de x_2, pois este é determinado como $x_2 = C - x_1$.

```
1  int main() {
2      int C, x1, x2;
3
4      printf("Digite o valor de C: ");
5      scanf("%d", &C);
6
7      for (x1 = 0; x1 <= C; x1++) {
8          x2 = C - x1;
9          printf("%d + %d = %d\n", x1, x2, C);
10     }
11
12     return 0;
13 }
```

Equações lineares inteiras

Problema

Quais são as soluções de $x_1 + x_2 + x_3 = C$ com $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, $C \geq 0$ e todas inteiras?

Equações lineares inteiras

Uma solução: para cada um dos valores de x_1, com $0 \leq x_1 \leq C$, teste todos os valores de x_2 e x_3 e verifique quais deles são soluções.

```
1  para cada x1 entre 0 e C faça:
2      para cada x2 entre 0 e C faça:
3          para cada x3 entre 0 e C faça:
4              se x1 + x2 + x3 = C então imprima solução
```

Equações lineares inteiras

```c
int main() {
    int C, x1, x2, x3;

    printf("Digite o valor de C: ");
    scanf("%d", &C);

    for (x1 = 0; x1 <= C; x1++) {
        for (x2 = 0; x2 <= C; x2++) {
            for (x3 = 0; x3 <= C; x3++) {
                if (x1 + x2 + x3 == C)
                    printf("%d + %d + %d = %d\n", x1, x2, x3, C);
            }
        }
    }

    return 0;
}
```

Equações lineares inteiras

- Note que, fixado x_1, o valor máximo de x_2 é $C - x_1$.
- Fixados x_1 e x_2, o valor de x_3 é determinado como $C - x_1 - x_2$.
- Podemos alterar o programa com estas melhorias.

```
1  int main() {
2      int C, x1, x2, x3;
3
4      printf("Digite o valor de C: ");
5      scanf("%d", &C);
6
7      for (x1 = 0; x1 <= C; x1++) {
8          for (x2 = 0; x2 <= C - x1; x2++) {
9              x3 = C - x1 - x2;
10             printf("%d + %d + %d = %d\n",x1, x2, x3, C);
11         }
12     }
13
14     return 0;
15 }
```

Mega-Sena

Na Mega-Sena, um jogo consiste de 6 números distintos com valores entre 1 e 60.

Problema

Imprimir todos os jogos possíveis da Mega-Sena.

Mega-Sena

Partimos da mesma idéia dos dados: gerar todos os possíveis valores para cada um dos 6 números do jogo.

```
1   int main() {
2       int d1, d2, d3, d4, d5, d6;
3
4       for (d1 = 1; d1 <= 60; d1++)
5           for (d2 = 1; d2 <= 60; d2++)
6               for (d3 = 1; d3 <= 60; d3++)
7                   for (d4 = 1; d4 <= 60; d4++)
8                       for (d5 = 1; d5 <= 60; d5++)
9                           for (d6 = 1; d6 <= 60; d6++)
10                              printf("%d, %d, %d, %d, %d, %d\n", d1, d2, d3, d4, d5, d6);
11
12       return 0;
13  }
```

Qual a saída deste programa? Ele está correto?

Mega-Sena

```c
int main() {
    int d1, d2, d3, d4, d5, d6;

    for (d1 = 1; d1 <= 60; d1++)
        for (d2 = 1; d2 <= 60; d2++)
            for (d3 = 1; d3 <= 60; d3++)
                for (d4 = 1; d4 <= 60; d4++)
                    for (d5 = 1; d5 <= 60; d5++)
                        for (d6 = 1; d6 <= 60; d6++)
                            printf("%d, %d, %d, %d, %d, %d\n", d1, d2, d3, d4, d5, d6);

     return 0;
 }
```

As primeiras linhas impressas por este programa serão:

```
1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 2
1, 1, 1, 1, 1, 3
1, 1, 1, 1, 1, 4
1, 1, 1, 1, 1, 5
1, 1, 1, 1, 1, 6
1, 1, 1, 1, 1, 7
1, 1, 1, 1, 1, 8
1, 1, 1, 1, 1, 9
```

Mega-Sena

O programa anterior repete números, portanto devemos remover repetições.

```
1   int main() {
2       int d1, d2, d3, d4, d5, d6;
3
4       for (d1 = 1; d1 <= 60; d1++)
5           for (d2 = 1; d2 <= 60; d2++)
6               for (d3 = 1; d3 <= 60; d3++)
7                   for (d4 = 1; d4 <= 60; d4++)
8                       for (d5 = 1; d5 <= 60; d5++)
9                           for (d6 = 1; d6 <= 60; d6++)
10                              if ((d1 != d2) && (d1 != d3) && ...)
11                                  printf("%d, %d, %d, %d, %d, %d\n", d1, d2, d3, d4, d5, d6);
12
13       return 0;
14  }
```

Após incluir todos os testes para garantir que os números são distintos, temos a solução?

Mega-Sena

- Não temos uma solução válida, pois o programa irá imprimir jogos como:

```
1  12, 34, 8, 19, 4, 45
2  34, 12, 8, 19, 4, 45
3  34, 12, 19, 8, 4, 45
```

- Todos estes são um único jogo: 4, 8, 12, 19, 34, 45.
- Podemos assumir então que um jogo é sempre apresentado com os números em ordem crescente.
- Dado que fixamos o valor de **d1**, **d2** necessariamente é maior que **d1**.
- Após fixar **d1** e **d2**, **d3** deve ser maior que **d2**, e etc.

Mega-Sena

Solução correta:

```c
int main() {
    int d1, d2, d3, d4, d5, d6;

    for (d1 = 1; d1 <= 60; d1++)
        for (d2 = d1 + 1; d2 <= 60; d2++)
            for (d3 = d2 + 1; d3 <= 60; d3++)
                for (d4 = d3 + 1; d4 <= 60; d4++)
                    for (d5 = d4 + 1; d5 <= 60; d5++)
                        for (d6 = d5 + 1; d6 <= 60; d6++)
                            printf("%d, %d, %d, %d, %d, %d\n", d1, d2, d3, d4, d5, d6);

    return 0;
}
```

Números em ordem

Problema

Fazer um programa que lê n números inteiros do teclado, e no final informa se os números lidos estão ou não em ordem crescente.

Números em ordem

- Um laço principal será responsável pela leitura dos números.
- Vamos usar duas variáveis, uma que guarda o número lido na iteração atual, e uma que guarda o número lido na iteração anterior.
- Os números estarão ordenados se a condição (anterior <= atual) for válida durante a leitura de todos os números.

```
1  leia um número e salve em n
2  ordenado = 1 /* Assumimos que os números estão ordenados */
3  leia um número e salve em anterior
4  repita (n-1) vezes:
5      leia um número e salve em atual
6      se atual < anterior
7          ordenado = 0
8      anterior = atual
```

Números em ordem

```c
#include <stdio.h>

int main() {
    int i, n, atual, anterior, ordenado;

    printf("Digite o valor de n:");
    scanf("%d", &n);

    scanf("%d", &anterior);
    i = 1; /* já leu um número */

    ordenado = 1;
    while (i < n && ordenado) {
        scanf("%d", &atual);
        i++;
        if (atual < anterior)
            ordenado = 0;
        anterior = atual;
    }

    if (ordenado)
        printf("Sequência ordenada!\n");
    else
        printf("Sequência não ordenada!\n");

    return 0;
}
```

Números de Fibonacci

- A série de Fibonacci é: $1, 1, 2, 3, 5, 8, 13, \ldots$
- Ou seja, o n-ésimo termo é a soma dos dois termos anteriores

$$F(n) = F(n-1) + F(n-2) \ ,$$

onde $F(1) = 1$ e $F(2) = 1$.

Problema

Fazer um programa que imprime os primeiros n números da série de Fibonacci.

Números de Fibonacci

```
1  leia um número e salve em n
2  contador = 1
3  f_atual = 1, f_ant = 0
4  enquanto contador <= n faça
5      imprima f_atual
6      aux = f_atual
7      f_atual = f_atual + f_ant
8      f_ant = aux
9      contador = contador +1
```

Números de Fibonacci

```c
int main() {
    int n, f_ant, f_atual, f_aux, cont;

    printf("Digite um número:");
    scanf("%d", &n);

    cont = 1;
    f_ant = 0;
    f_atual = 1;
    while (cont <= n) {
        printf("%d, ", f_atual);
        f_aux = f_atual;
        f_atual = f_atual + f_ant;
        f_ant = f_aux;
        cont++;
    }
    printf("\n");

    return 0;
}
```

Menu de escolhas

- Em programas de computador, é comum a apresentação de um menu de opções para o usuário.
- Vamos fazer um menu com algumas opções, incluindo uma última para encerrar o programa.

Menu de escolhas

O programa terá as seguintes opções:

- **1** - Cadastrar um produto.
- **2** - Buscar informações de produto.
- **3** - Remover um produto.
- **4** - Sair do Programa.

Após realizar uma das operações, o programa volta para o menu.

Menu de escolhas

O comportamento do seu programa deveria ser algo como:

```c
do {
    printf("1 - Cadastrar um produto\n");
    printf("2 - Buscar informações de produto\n");
    printf("3 - Remover um produto\n");
    printf("4 - Sair do programa\n");
    printf("Entre com a opção: ");
    scanf("%d", &opcao);

    /* Faça o que for esperado conforme opção digitada */

} while (opcao != 4);
```

Menu de escolhas

```c
int main() {
    int opcao;

    do {
        printf("1 - Cadastrar um produto\n");
        printf("2 - Buscar informações de produto\n");
        printf("3 - Remover um produto\n");
        printf("4 - Sair do programa\n");
        printf("Entre com a opção: ");
        scanf("%d", &opcao);

        if (opcao == 1)
            printf("Cadastrando....\n\n\n");
        else if (opcao == 2)
            printf("Buscando......\n\n\n");
        else if (opcao == 3)
            printf("Removendo.....\n\n\n");
        else if (opcao == 4)
            printf("Seu programa será encerrado.\n\n\n");
        else
            printf("Opção Inválida!\n\n\n");
    } while (opcao != 4);

    return 0;
}
```

Calculando potências de 2

Problema

Imprimir as potências $2^0, 2^1, \ldots, 2^n$ para um n qualquer.

Calculando potências de 2

- Usamos uma variável acumuladora que no início da i-ésima iteração de um laço, possui o valor 2^i.
- Imprimimos este valor e atualizamos a acumuladora para a próxima iteração, multiplicando esta variável por 2.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem o valor de 2^i que é impresso.
  - No fim da i-ésima iteração seu valor é atualizado para 2^{i+1} para a próxima iteração.

```
1  acumuladora = 1 /* Corresponde a 2^0 */
2  para i = 0 até n faça:
3      imprima acumuladora
4      acumuladora = acumuladora * 2
```

Calculando potências de 2

A solução pode ser obtida utilizando-se o laço **for**.

```
1  pot = 1; /* corresponde a 2^0 */
2  for (i = 0; i <= n; i++) {
3      printf("%d\n", pot);
4      pot = pot * 2;
5  }
```

Calculando potências de 2

Também pode ser obtida utilizando o comando **while**.

```
1  int i, n, pot;
2
3  scanf("%d", &n);
4
5  pot = 1;
6  i = 0;
7  while (i <= n) {
8      printf("2^%d = %d\n", i, pot);
9      pot = pot *2;
10     i++;
11 }
```

Representação binário-decimal

Já sabemos que um computador armazena todas as informações na representação binária.

É útil saber como converter valores binários em decimais e vice versa.

Problema

Dado um número em binário, encontrar o seu correspondente em decimal.

Representação binário-decimal

- Dado um número em binário $b_n b_{n-1} \ldots b_2 b_1 b_0$, este corresponde na forma decimal a:

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

- Exemplos:

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

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

- OBS: Em uma palavra no computador, um bit é usado para indicar o sinal do número: $-$ ou $+$.

Representação binário-decimal

- Seja o número 10101 em binário.
- Qual o seu valor em decimal?

Representação binário-decimal

- Seja o número 10101 em binário.
- Qual o seu valor em decimal?
- **Resposta:** $21 = 2^4 + 2^2 + 2^0$

Representação binário-decimal

- Vamos supor que lemos do teclado um inteiro em 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)?
- Basta usarmos a expressão:

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

Representação binário-decimal

Um passo importante é conseguir recuperar os dígitos individuais do número:

- Note que $n\%10$ recupera o último dígito de n.
- Note que $n/10$ remove o último dígito de n, pois ocorre a divisão inteira por 10.

Exemplo: Com $n = 345$, ao fazermos $n\%10$ obtemos 5. E ao fazermos $n/10$ obtemos 34.

Representação binário-decimal

Para obter cada um dos dígitos de um número n podemos fazer algo como:

```
1  leia n
2  enquanto n != 0 faça:
3      digito = n % 10
4      imprima o digito
5      n = n/10
```

Representação binário-decimal

O programa abaixo imprime cada um dos dígitos de n:

```
1  int main() {
2      int n, digito;
3
4      printf("\n Digite um número:");
5      scanf("%d", &n);
6
7      while (n != 0) {
8          digito = n % 10;
9          printf("%d\n", digito);
10         n = n/10;
11     }
12
13     return 0;
14 }
```

Representação binário-decimal

- Usar a fórmula $\sum_{i=0}^{n} b_i \times 2^i$ para transformar um número em binário para decimal.
- Devemos gerar as potências $2^0, \ldots, 2^n$ e multiplicar cada potência 2^i pelo i-ésimo dígito.
    - Calcular as potências já sabemos (acumuladora **pot**).
- Para armazenar a soma $\sum_{i=0}^{n} b_i \times 2^i$, usamos uma outra variável acumuladora **soma**.

Representação binário-decimal

```
1  leia n
2  pot = 1
3  soma = 0
4  enquanto n != 0 faça:
5      digito = n % 10
6      n = n/10
7      soma = soma + (pot*digito)
8      pot = pot * 2
```

Representação binário-decimal

```c
int main() {
    int n, digito, soma, pot;

    printf("Digite um número em binário: ");
    scanf("%d", &n);

    soma = 0;
    pot = 1;
    while (n != 0) {
        digito = n % 10;
        n = n/10;
        soma = soma + (digito*pot);
        pot = pot*2;
    }
    printf("Valor em decimal: %d\n", soma);

    return 0;
}
```

Representação decimal-binário

Problema

Dado um número em decimal, encontrar o seu correspondente em binário.

Representação decimal-binário

- Qualquer decimal pode ser escrito como uma soma de potências de 2: $5 = 2^2 + 2^0$ e $13 = 2^3 + 2^2 + 2^0$, por exemplo.
- Nesta soma, para cada potência 2^i, sabemos que na representação em binário haverá um 1 no i-ésimo dígito. Exemplo: $13 = 1101$.
- O que acontece se fizermos sucessivas divisões por 2 de um número decimal?

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

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

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

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

Representação decimal-binário

- Dado n em decimal, fazemos repetidas divisões por 2, obtendo os dígitos do valor em binário:

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

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

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

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

```
1  leia n
2  enquanto n != 0 faça:
3      digito = n % 2
4      imprima digito
5      n = n/2
```

Representação decimal-binário

```
1  int main() {
2      int n, digito;
3
4      printf("Digite um número:");
5      scanf("%d", &n);
6
7      while (n != 0) {
8          digito = n % 2;
9          n = n/2;
10         printf("%d\n", digito);
11     }
12
13     return 0;
14 }
```

Dados

Problema

Imprimir todas as possibilidades de resultados ao se jogar 4 dados de 6 faces.

Dados

- Para cada possibilidade do primeiro dado, devemos imprimir todas as possibilidades dos 3 dados restantes.
- Para cada possibilidade do primeiro e segundo dados, devemos imprimir todas as possibilidades dos 2 dados restantes.
- ...
- Você consegue pensar em uma solução com laços aninhados?

Dados

```c
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;
}
```

Calculando o valor de $n!$

Problema

Fazer um programa que lê um valor inteiro positivo n e calcula o valor de $n!$.

Lembre-se que $n! = n \times (n-1) \times (n-2) \times \ldots 2 \times 1$.

Calculando o valor de $n!$

- Criamos uma variável acumuladora que no início da i-ésima iteração de um laço armazena o valor de $(i-1)!$.
- Durante a i-ésima iteração atualizamos a variável acumuladora multiplicando esta por i obtendo $i!$.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem o valor de $(i-1)!$.
  - No fim da i-ésima iteração seu valor é atualizado para $i! = (i-1)! \times i$.
- No fim do laço, após n iterações, teremos na acumuladora o valor de $n!$.

```
1  acumuladora = 1 /* corresponde a 0! */
2  para i = 1 até n faça:
3      acumuladora = acumuladora * i
4      i = i + 1
```

Calculando o valor de $n!$

```c
#include <stdio.h>

int main() {
    int i, n, fat;

    scanf("%d", &n);

    for(fat = 1, i = 1; i <= n; i++) {
        fat = fat * i;
    }

    printf("Fatorial de %d e: %d\n", n, fat);

    return 0;
}
```

Números primos: outra solução

- Um número n é primo se nenhum número de 2 até $(n-1)$ dividi-lo.
- Podemos usar uma variável que conta quantos números dividem n.
- Se o número de divisores for 0, então n é primo.

```
1  leia um número e salve em n
2  div = 2
3  divisores = 0 /* ninguém divide n ainda */
4  enquanto div <= (n-1) faça
5      se (n % div) == 0
6          divisores = divisores + 1
7      div = div + 1
8  se divisores == 0 então
9      número é primo
```

Números primos

```c
int main() {
    int div, n, divisores;
    printf("Digite um número:");
    scanf("%d", &n);

    div = 2;
    divisores = 0;
    while (div <= n-1) {
        if (n % div == 0)
            divisores++;
        div++;
    }

    if (divisores == 0)
        printf("É primo!\n");
    else
        printf("Não é primo!\n");

    return 0;
}
```

Números primos

É claro que é melhor terminar o laço assim que descobrirmos algum divisor de n.

```
1   int main() {
2       int div, n, divisores;
3
4       printf("Digite um numero:");
5       scanf("%d", &n);
6
7       div = 2;
8       divisores = 0;
9       while (div <= n-1 && divisores == 0) {
10          if (n % div == 0)
11              divisores++;
12          div++;
13      }
14
15      if (divisores == 0)
16          printf("É primo!\n");
17      else
18          printf("Não é primo!\n");
19
20      return 0;
21  }
```

Exercícios

Exercício

Faça um programa que imprima um menu de 4 pratos na tela e uma quinta opção para sair do programa.

O programa deve imprimir o prato solicitado.

O programa deve terminar quando for escolhida a quinta opção.

Exercício

Faça um programa que lê dois números inteiros positivos a e b.

Utilizando laços, o seu programa deve calcular e imprimir o valor a^b.

Exercício

Faça um programa que lê um número n e que computa e imprima o valor

$$\sum_{i=1}^{n} i \ .$$

OBS: Não use fórmulas como a da soma de uma P.A.

Exercício

Faça um programa que lê um número n e imprima os valores entre 2 e n que são divisores de n.

Exercício

Faça um programa que lê um número n e imprima os valores

$$\sum_{i=1}^{j} i$$

para j variando de 1 até n, um valor por linha.

Exercício

No exemplo dos números primos, não precisamos testar todos os números entre $2, \ldots, (n-1)$, para verificar se dividem ou não n.

Basta testarmos até $n/2$.

Por quê? Qual o maior divisor possível de n?

Na verdade, basta testarmos os números $2, \ldots, \sqrt{n}$.

Por quê?

Exercício

Considere o programa para determinar se uma sequência de n números digitados pelo usuário está ordenada ou não.

Refaça o programa usando uma variável contadora ao invés de indicadora.

Exercício

Faça um programa em C que calcule o máximo divisor comum (MDC) de dois números m e n.

Você deve utilizar a seguinte regra do cálculo do MDC, com $m \geq n$:

$$mdc(m, n) = m \text{ se } n = 0$$

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

Exercício

Na transformação de decimal para binário, modifique o programa para que este guarde o valor binário em uma variável inteira ao invés de imprimir os dígitos um por linha na tela.

Dica: Suponha que $n = 7$ (111 em binário), e que você já computou $x = 11$. Para "inserir" o último dígito 1 em x você deve fazer $x = x + 100$. Ou seja, você precisa de uma variável acumuladora que armazena as potências de 10: 1, 10, 100, 1000 etc.

Exercício

Implemente um programa que compute todas as soluções de equações do tipo

$$x_1 + x_2 + x_3 + x_4 = C$$

Melhore o seu programa com as seguinte idéias:

- Fixado x_1, os valores possíveis para x_2 são $0, \ldots, C - x_1$.
- Fixado x_1 e x_2, os valores possíveis para x_3 são $0, \ldots, C - x_1 - x_2$.
- Fixados x_1, x_2 e x_3, então x_4 é unicamente determinado.

Exercício

Faça um programa que leia um número n e imprima n linhas na tela com o seguinte formato (exemplo se $n = 6$):

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

Exercício

Faça um programa que leia um número n e imprima n linhas na tela com o seguinte formato (exemplo se $n = 6$):

```
1  + * * * * *
2  * + * * * *
3  * * + * * *
4  * * * + * *
5  * * * * + *
6  * * * * * +
```

Exercício

Um jogador da Mega-Sena é supersticioso e só faz jogos em que o primeiro número do jogo é par, o segundo é ímpar, o terceiro é par, o quarto é ímpar, o quinto é par e o sexto é ímpar.

Faça um programa que imprima todas as possibilidades de jogos que este jogador supersticioso pode jogar.