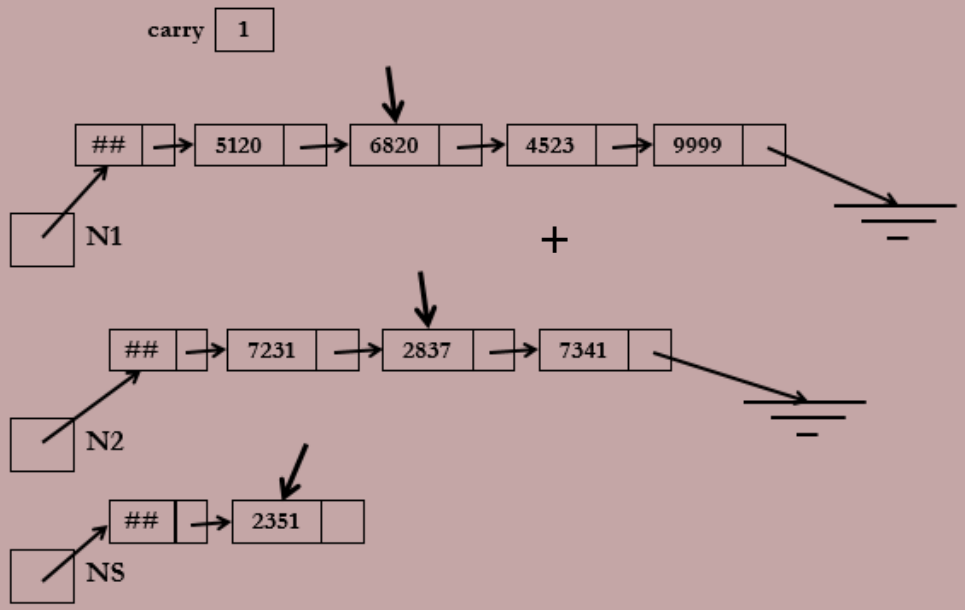
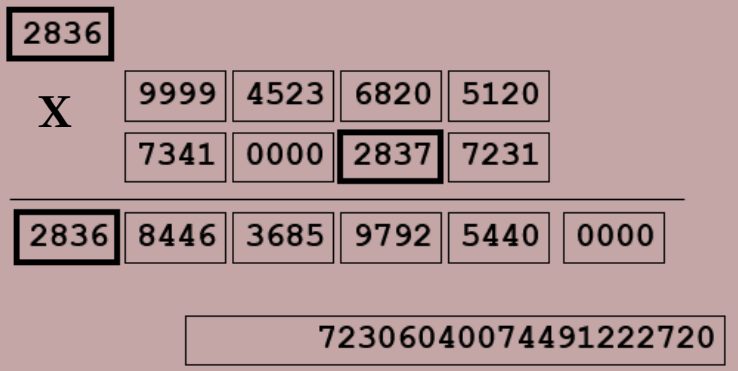
AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Fábio Carneiro Mokarzel
Nei Yoshihiro Soma





Esta obra pretende servir como um guia para a preparação de aulas práticas de disciplinas introdutórias de Computação e Programação, em cursos de Engenharia.
Previamente, os autores lançaram um conjunto de três livros que se constituem numa espécie de manual de ensino dessas disciplinas. Esta obra vem complementar esse conjunto com propostas de atividades práticas de programação.
Essas atividades são apresentadas de forma progressiva em termos de dificuldade. Cada capítulo começa com programas e exercícios elementares. Seu grau de complexidade vai aumentando e, no final, é proposto um exercício laboratorial abrangendo vários dos aspectos cobertos pelos programas e exercícios anteriores.
A obra visa dar ao professor um conjunto de exercícios que proporcione aos seus alunos substancial experiência, abordando vários aspectos de programação unimodular e multimodular.

Aulas Práticas de Introdução à Programação

Fábio Carneiro Mokarzel

Nei Yoshihiro Soma

Aulas Práticas de Introdução à Programação

Fábio Carneiro Mokarzel

Nei Yoshihiro Soma



Reservado para ISBN

FÁBIO CARNEIRO MOKARZEL

Disse o Senhor Deus, Criador e Sustentador dos céus e de todo o universo: "Buscar-me-eis e me achareis, quando me buscardes de todo o vosso coração"

Dedico este trabalho ao Senhor Jesus Cristo, Meu Criador, Salvador e Libertador de minha alma e à minha esposa Ilda Alves dos Santos Mokarzel

NEI YOSHIHIRO SOMA

Não tenho compromisso com minhas ideias, pois o único compromisso que se pode ter em termos de ideias é com a busca da verdade **(Prof. Anísio Teixeira)**

A meus pais, Yoshio e Verônica (*in memoriam*), irmã Nanci, esposa Akemi e filhos Renato, Ryuji e Daniel, na ordem em que apareceram na minha vida.

Os autores também dedicam esta obra aos professores que vierem a abraçar a responsabilidade de ministrar disciplinas introdutórias sobre programação de computadores.

Agradecimentos

Às nossas esposas, Ilda e Akemi, pelas incessantes orações, apoio, incentivo e constantes exortações para que perseverássemos na elaboração deste livro.

A todos os alunos do ITA para os quais ministramos as disciplinas de Introdução à Computação e à Programação, pelo retorno que nos proporcionaram sobre o material didático que lhes temos apresentado durante as diversas etapas de sua produção.

Prefácio

Desde os Anos 80, quando, no Instituto Tecnológico de Aeronáutica, iniciamos nossa carreira como Professores Universitários, vimos ministrando, entre outras disciplinas de Ciência e Engenharia de Computação, disciplinas introdutórias de Computação e de Programação. Para nos auxiliar na transmissão dos conhecimentos sobre esses assuntos, vimos produzindo, modernizando e aperfeiçoando material didático para apresentar aos alunos em classe e deixa-lo disponível para estudos extra classe.

Como fruto dessa experiência de ensino, produzimos dois livros textos sobre o assunto, sendo o segundo uma revisão e aperfeiçoamento do primeiro, e mais um conjunto de três livros contendo orientações para o preparo de aulas teóricas dessas disciplinas, visando atender as necessidades de professores que vierem a ministra-las pela primeira vez.

Para complementar esse conjunto de três livros, estamos lançando esta obra, cujo objetivo é servir de orientação para o preparo de aulas práticas das referidas disciplinas.

Em nossa carreira, tivemos também a oportunidade de desenvolver roteiros para aulas nas quais levamos nossos alunos para uma sala de computadores e lhes ensinamos os primeiros passos para a produção de programas, num ambiente de software apropriado. Para tanto, utilizamos em cada aula uma abordagem progressiva, levando inicialmente os alunos a executarem programas e exercícios elementares, para, em seguida, aumentarmos a dificuldade desses exercícios, até chegarmos ao final, propondo um exercício laboratorial, englobando os temas de vários exercícios anteriores, ficando os alunos incumbidos de entrega-los para serem avaliados.

O presente trabalho é dividido em onze capítulos, cada um correspondendo a uma aula prática, para a qual deve ser desenvolvido um trabalho laboratorial a ser avaliado pelo professor.

Foi escolhida a Linguagem C para a realização de todos os programas e exercícios deste livro e o ambiente de programação utilizado é o Code Blocks.

RESUMO DO CONTEÚDO DOS CAPÍTULOS

O Capítulo I, *Comandos Simples e Repetições*, começa orientando os alunos a preparar o ambiente de programação e o local a ser usado para armazenar os programas a serem desenvolvidos. Em seguida, o capítulo apresenta programas para escrever frases simples, repetições dessas frases, programas com variáveis não declaradas e programas amigáveis destinados a escrever valores de variáveis. No final, é proposta a elaboração de um programa para ler a compra de diversos materiais escolares por parte de vários alunos, calculando o valor gasto por eles.

O Capítulo II, *Confecção de Tabelas*, começa ensinando os alunos a numerar as linhas de diversas frases simples repetidas. Em seguida, o capítulo apresenta programas para alinhar verticalmente a numeração e as frases escritas, usando funções de escrita com formatação, e para separar por barra vertical a numeração e as frases repetidas, obtendo assim uma tabela com duas colunas, inclusive provida de moldura. No final, é proposta a elaboração de um programa para fazer uma única tabela de três funções matemáticas.

O Capítulo III, *Comandos de Controle*, começa apresentando programas bem simples envolvendo comandos condicionais e repetitivos. Em seguida, vêm programas para testar divisibilidade entre números inteiros e, no final, é proposta a elaboração de um programa que leia uma sequência de números inteiros e faça um relatório sobre eles, reportando sua média aritmética, a soma de seus números pares, a soma de seus números positivos múltiplos de 3 e de 7, o maior e o menor dentre eles e os números lidos que são fatoriais e os que pertencem à sequência de Fibonacci.

O Capítulo IV, *Fluxogramas*, apresenta inicialmente um fluxograma para calcular as raízes de uma equação do segundo grau e a obtenção de um programa em C equivalente a ele. Em seguida, propõe exercícios de obtenção de programas a partir de outros fluxogramas, dois dos quais devendo ser avaliados por professores.

O Capítulo V, *Comandos For e Switch-Case*, mostra inicialmente, através de um programa, a correspondência entre os comandos *For* e *While*. Em seguida, apresenta dois programas ilustrativos do comando *Switch-Case*. Depois, o capítulo mostra a implementação de um autômato finito, usando este último comando, e, no final, propõe a implementação de três outros autômatos finitos, espelhando-se no exemplo anterior.

O Capítulo VI, *Desenhos e Animação*, mostra inicialmente funções para o controle do cursor do vídeo-texto, usando dois programas ilustrativos. Em seguida, aborda o tema de animação no vídeo-texto, ilustrando com um programa para movimentar um asterisco pela tela. No final, propõe um programa para fazer um desenho animado de tema livre, usando os artifícios mostrados anteriormente.

O Capítulo VII, *Variáveis Indexadas Numéricas*, começa apresentando programas bem simples envolvendo vetores e matrizes numéricas. Em seguida, vêm programas para ler e escrever matrizes usando teclado, vídeo e arquivos em disco. No final, o capítulo propõe um programa para ler uma matriz de um arquivo e escreve-la em outro arquivo, juntamente com vários vetores obtidos a partir dela.

O Capítulo VIII, *Cadeias de Caracteres e Estruturas*, apresenta inicialmente programas para leitura e escrita de cadeias de caracteres, usando dispositivos da Linguagem C para sua manipulação. Em seguida, o capítulo aborda o tema de conversão de bases genéricas para a base decimal, usando estruturas para armazenar os dígitos e as bases dos números. No final, é proposta a elaboração de um programa para ler vários números em bases diversas e calcular sua somatória na base decimal.

O Capítulo IX, *Estruturas e Subprogramas sem Parâmetros*, desenvolve o esqueleto de um programa destinado a emitir um relatório sobre um campeonato de pontos corridos em dois turnos. As informações sobre cada time são guardadas numa estrutura. São elas o nome do time, sua pontuação, número de vitórias, gols marcados e sofridos e saldo de gols. Os resultados dos jogos são guardados numa matriz. O relatório deve fazer uma classificação do campeonato, usando essa matriz para calcular o desempenho dos times. O trabalho a ser entregue pelos alunos é o de completar o referido programa.

O Capítulo X, *Subprogramas com Parâmetros e Recursividade*, apresenta vários programas e exercícios contendo subprogramas recursivos e, no final, propõe um programa para o cálculo de determinantes, usando a Regra de Laplace, cuja formulação é recursiva.

Finalmente o Capítulo XI, *Encadeamento de Estruturas por Ponteiros*, mostra inicialmente como se obter um encadeamento de estruturas, usando ponteiros, e como escrever o seu conteúdo. Essa apresentação inclui a modularização de um programa, com um subprograma específico para formar o encadeamento mediante a leitura de seus elementos e outro específico para escrever o conteúdo de suas estruturas. No final, o capítulo propõe um programa para armazenar, somar e multiplicar dois números decimais muito grandes a serem guardados cada um num encadeamento de estruturas. Tal programa deverá ser devidamente modularizado, com subprogramas específicos para armazenar, escrever, somar e multiplicar números guardados em tais encadeamentos.

Nossa expectativa com esta obra é a de que professores que venham a ministrar as referidas disciplinas possam ter um roteiro seguro para elaborar suas aulas práticas, proporcionando aos seus alunos substancial experiência de programação.

Sumário

Dedicatória 7

Agradecimentos 9

Prefácio 11

Sumário 16

Capítulo I – Comandos Simples e Repetições 17

Capítulo II – Confecção de Tabelas 44

Capítulo III – Comandos de Controle 80

Capítulo IV – Fluxogramas 114

Capítulo V – Comandos For e Switch-Case 129

Capítulo VI – Desenhos e Animação 155

Capítulo VII – Variáveis Indexadas Numéricas 168

Capítulo VIII – Cadeias de Caracteres e Estruturas 194

Capítulo IX – Estruturas e Subprogramas sem Parâmetros 225

Capítulo X – Subprogramas com Parâmetros e Recursividade 249

Capítulo XI – Encadeamento de Estruturas por Ponteiros 293

Bibliografia 461

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo I

Comandos Simples e Repetições

Preparo do Ambiente:

- Criar no diretório **"Documentos"** uma pasta com o nome da disciplina e nela criar a subpasta **"Programas"**

- Por questões de **organização pessoal**, colocar nessa pasta todos os arquivos criados nas **atividades de laboratório** dessa disciplina

- Cada **programa digitado** será guardado num **arquivo** de extensão **.cpp**

- A **compilação** e **execução** de um programa gera **dois arquivos** com o mesmo nome, mas com **extensões diferentes** (**.o**: programa objeto e **.exe**: programa executável)

- **Programa 1.1: Comandos de saída de resultados**

- Abrir o **CodeBlocks**

- Clicar em **File/New/Empty file**

- **Digitar** o programa a seguir

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    return 0;
}
```

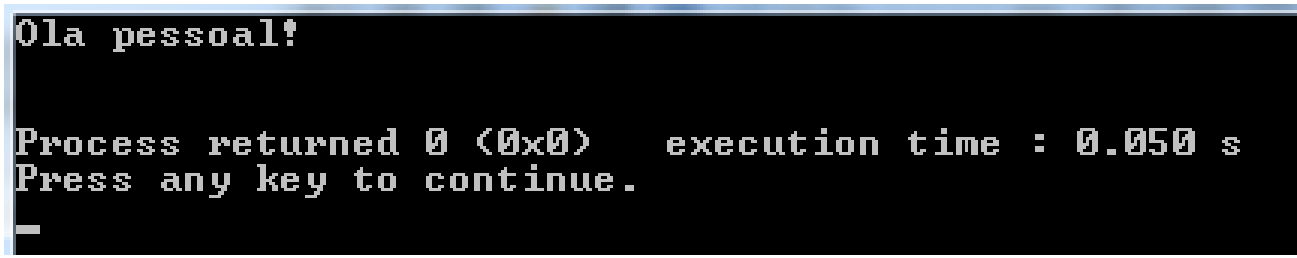
- Clicar em **File** e salvar como **"Saida.cpp"** na pasta **"Programas"**

- Clicar no botão  **(build and run**) para compilar e executar

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    return 0;
}
```

Resultado na tela:



- **printf ("Ola pessoal!")**: escreve no vídeo o que estiver entre aspas **(“- - - - -”)**, exceto as aspas

- **printf ("\n\n")**: não escreve os dois **“\n”s**; para cada **“\n”**, o cursor do vídeo vai para o início da linha seguinte

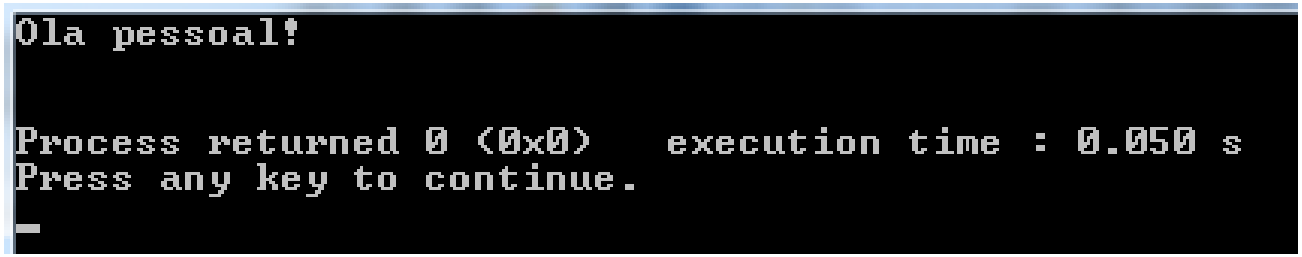
- Examinar, no **Windows Explorer**, a pasta **"Programas"** e constatar a geração de dois outros arquivos:

  - **Saida.o:** programa objeto gerado pelo Assembler
  - **Saida.exe:** programa executável (ou, simplesmente, **Saida**, de tipo **Aplicativo**

- Agora, clicar duas vezes em **Saida.exe**

- A **tela** de execução **abre**, escreve **"Ola pessoal!"** e **fecha** em seguida, não possibilitando a **visualização** do resultado

- Rodando dentro do **CodeBlocks**:



- A execução do programa se encerra e o **CodeBlocks** emite a mensagem:

  Process returned 0 (0x0) execution time : 0.050 s

  Press any key to continue

- Rodando fora do **CodeBlocks**, a execução do programa se encerra e a tela de execução é apagada

- Alterando o programa, para a tela não apagar:

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

```
Ola pessoal!

Pressione qualquer tecla para continuar. . .

Process returned 0 (0x0)   execution time : 16.930 s
Press any key to continue.
```

Resultado na tela

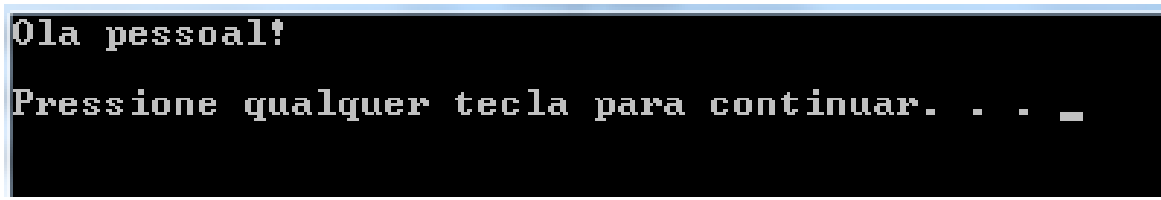
<---------- (system ("pause");)

- Clicar em **Save** e clicar no botão  **(build and run)** para compilar e executar

- **system ("pause")**: emite a mensagem **"Pressione qualquer tecla para continuar …"** e fica esperando a digitação

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```



Resultado na tela

- Executar o programa fora do **CodeBlocks** (clicar duas vezes em **Saida.exe**)

- A tela de execução só apaga quando algo for digitado

- Alterar o programa para:

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

Numa linha, tudo após "//" se torna um "comentário"

Comentários são eliminados do programa-fonte pelo compilador

- Salvar e executar

- Alterar o programa para:

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

- Salvar e executar

Depois de acionada a execução, a mensagem de compilação (build messages):

warning: incompatible implicit declaration of built-in function 'printf' (enabled by default)

A função "printf" pertence ao arquivo "stdio.h" da biblioteca da Linguagem C

O comentário retirou a inclusão desse arquivo

Mesmo assim o compilador aceita o programa, mas avisa o ocorrido

- Alterar o programa para:

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

int main () {
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

- Salvar e executar

Há dois tipos de mensagens: warning e error

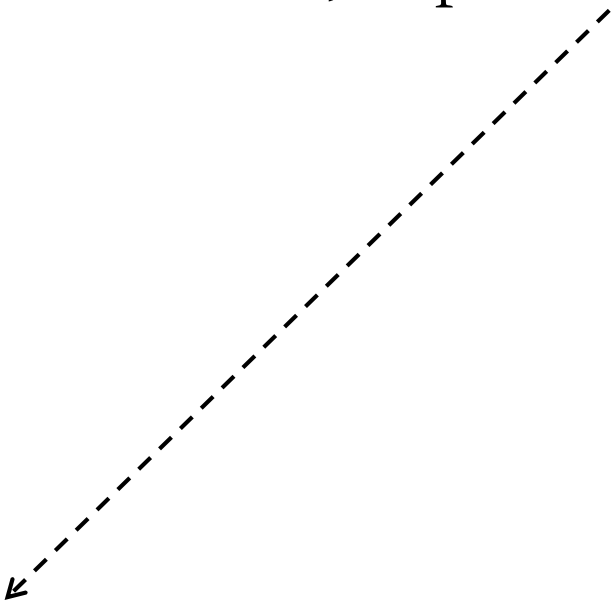
Warning: o compilador só avisa, mas faz a tradução

Error: o compilador avisa e não faz a tradução

A função "system" pertence ao arquivo "stdlib.h" da biblioteca da Linguagem C

A eliminação de seu "include" também provoca "warning"

- Voltando ao programa sem **“//”**, duplicando o primeiro **printf**:



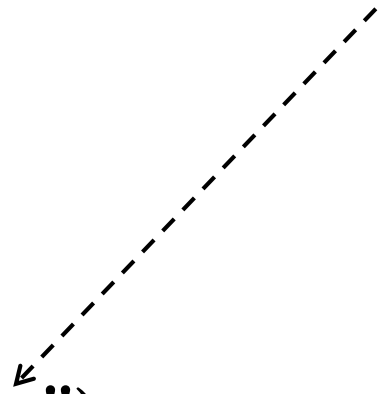
```
#include <stdio.h>
#include <stdlib.h>

int main () {
    printf ("Ola pessoal!");
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

- Salvar e executar

- Duas frases **“Ola pessoal!”** na mesma linha

- Alterar o programa para:



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

int main () {
    printf ("Ola pessoal!\n");
    printf ("Ola pessoal!");
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

- Salvar e executar

- A segunda frase **“Ola pessoal!”** na linha seguinte

Programa 1.2: Repetições de comandos de saída

- Clicar em **File/New/Empty file**
- **Digitar** o programa a seguir
- Salvá-lo como **"SaidaRepetida.cpp"** na pasta **"Programas"** e executar

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

int main () {
    i = 1;
    while (i <= 5) {
        printf ("Ola pessoal!\n");
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

Mensagem de erro de compilação:

Símbolo não declarado 'i'

É necessário declarar 'i'

- Alterar o programa para:

Salvar e executar

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

int main () {
    int i;
    i = 1;
    while (i <= 5) {
        printf ("Ola pessoal!\n");
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

int i: 'i' é uma variável inteira

Variável: nome ao qual está associado um local na memória, cujo valor pode ser alterado durante a execução

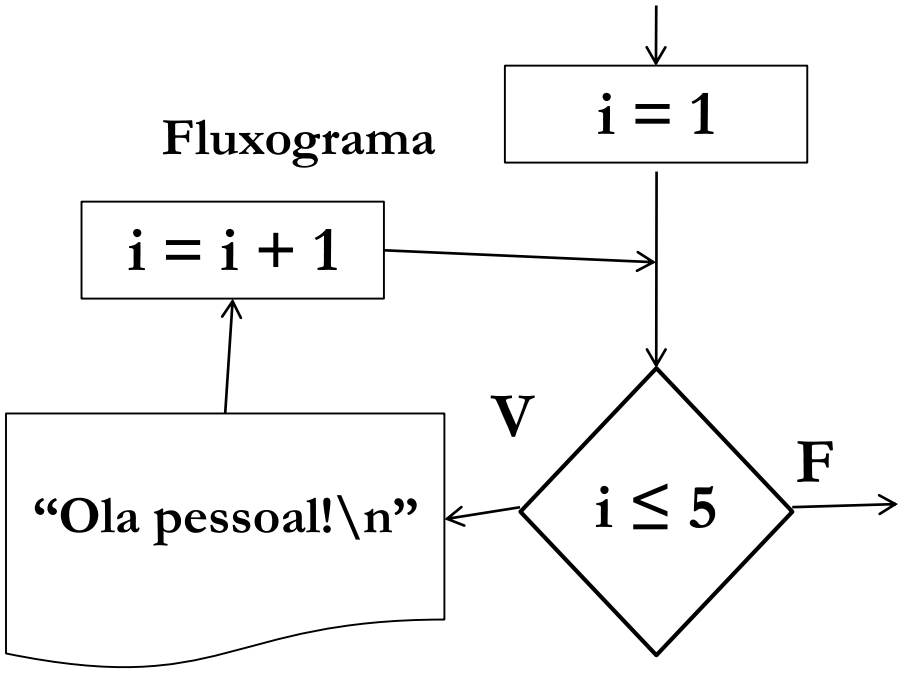
- Alterar o programa para:

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

int main () {
    int i;
    i = 1;
    while (i <= 5) {
        printf ("Ola pessoal!\n");
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

Funcionamento do while:

Fluxograma



Este programa só serve para escrever a frase 5 vezes

Que tal deixar o operador escolher o número de repetições?

Programa 1.3: Leitura amigável do número de repetições

- Clicar em **File/New/EmptyFile**
- **Digitar** o programa a seguir
- Salvá-lo como **"Leitura.cpp"** na pasta **"Programas"** e executar

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\nOla pessoal!");
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

Variável inteira 'n': destinada a guardar o número de repetições a ser digitado

Torna a leitura amigável: o operador saberá o que o programa fica esperando

Espera a digitação de um número inteiro (%d)

Lê o número digitado, guardando-o no endereço da variável 'n' (&n)

Programa 1.4: Escrita do valor de variáveis

- Quase sempre, é necessário escrever o valor de **variáveis** e de **expressões**
- A função **"printf"** tem artifícios para isso
- **Digitar**, salvar e executar o programa a seguir

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

int main () {
    int n; float preco;
    printf ("Numero de cadernos: ");
    scanf ("%d", &n);
    printf ("Preco do caderno: ");
    scanf ("%f", &preco);
    printf ("O preco de %d cadernos eh %g", n, n*preco);
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

float preco:

A variável 'preco' é do tipo real

Espera a digitação de um número real (%f)

Lê o número digitado, guardando-o no endereço da variável 'preco'

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

int main () {
    int n; float preco;
    printf ("Numero de cadernos: ");
    scanf ("%d", &n);
    printf ("Preco do caderno: ");
    scanf ("%f", &preco);
    printf ("O preco de %d cadernos eh %g", n, n*preco);
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

%d escreve o valor de 'n' (inteiro) em decimal

%g escreve o valor de 'n*preco' (real) em decimal, com o número mínimo necessário de casas após o ponto

Programa 1.5: Programa anterior com repetições

Digitar, salvar e executar este programa

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int aluno, numalunos, numcadernos; float preco;
    printf ("Digite o numero de alunos: ");
    scanf ("%d", &numalunos); printf ("\n"); aluno = 1;
    while (aluno <= numalunos) {
        printf ("Aluno %d: \n\n\tPreco do caderno: ", aluno);
        scanf ("%f", &preco);
        printf ("\tNumero de cadernos: "); scanf ("%d", &numcadernos);
        printf ("\tAluno %d: %d cadernos a R$%g cada; gasto total: R$%g\n\n",
                aluno, numcadernos, preco, numcadernos*preco);
        aluno = aluno + 1;
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

```
Digite o numero de alunos: 3

Aluno 1:

        Numero de cadernos: 5
        Preco do caderno: 7.3

        Numero de canetas: 6
        Preco da caneta: 2.6

        Numero de lapis: 8
        Preco do lapis: 1.3

        Numero de borrachas: 5
        Preco da borracha: 0.6

        Aluno 1
                5 cadernos a R$7.3 cada
                6 canetas a R$2.6 cada
                8 lapis a R$1.3 cada
                5 borrachas a R$0.6 cada

        Gasto total: R$65.5
```

1º Exercício Laboratorial: Alterar o programa anterior para que ele produza no vídeo resultados como este:

Em azul: digitados pelo operador

```
Aluno 2:

        Numero de cadernos: 7
        Preco do caderno: 6.2

        Numero de canetas: 8
        Preco da caneta: 3.1

        Numero de lapis: 6
        Preco do lapis: 1.3

        Numero de borrachas: 4
        Preco da borracha: 0.8

        Aluno 2
                7 cadernos a R$6.2 cada
                8 canetas a R$3.1 cada
                6 lapis a R$1.3 cada
                4 borrachas a R$0.8 cada

        Gasto total: R$79.2
```

1º Exercício Laboratorial: Alterar o programa anterior para que ele produza no vídeo resultados como este:

Em azul: digitados pelo operador

```
Aluno 3:

        Numero de cadernos: 9
        Preco do caderno: 8.3

        Numero de canetas: 3
        Preco da caneta: 2.7

        Numero de lapis: 4
        Preco do lapis: 1.7

        Numero de borrachas: 8
        Preco da borracha: 0.2

        Aluno 3
                9 cadernos a R$8.3 cada
                3 canetas a R$2.7 cada
                4 lapis a R$1.7 cada
                8 borrachas a R$0.2 cada

        Gasto total: R$91.2
```

1º Exercício Laboratorial: Alterar o programa anterior para que ele produza no vídeo resultados como este:

Em azul: digitados pelo operador

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo II

Confecção de Tabelas

Programa 2.1: Numeração das linhas

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%d Ola pessoal!", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```



Seja o programa a seguir, quase igual ao Programa 1.3

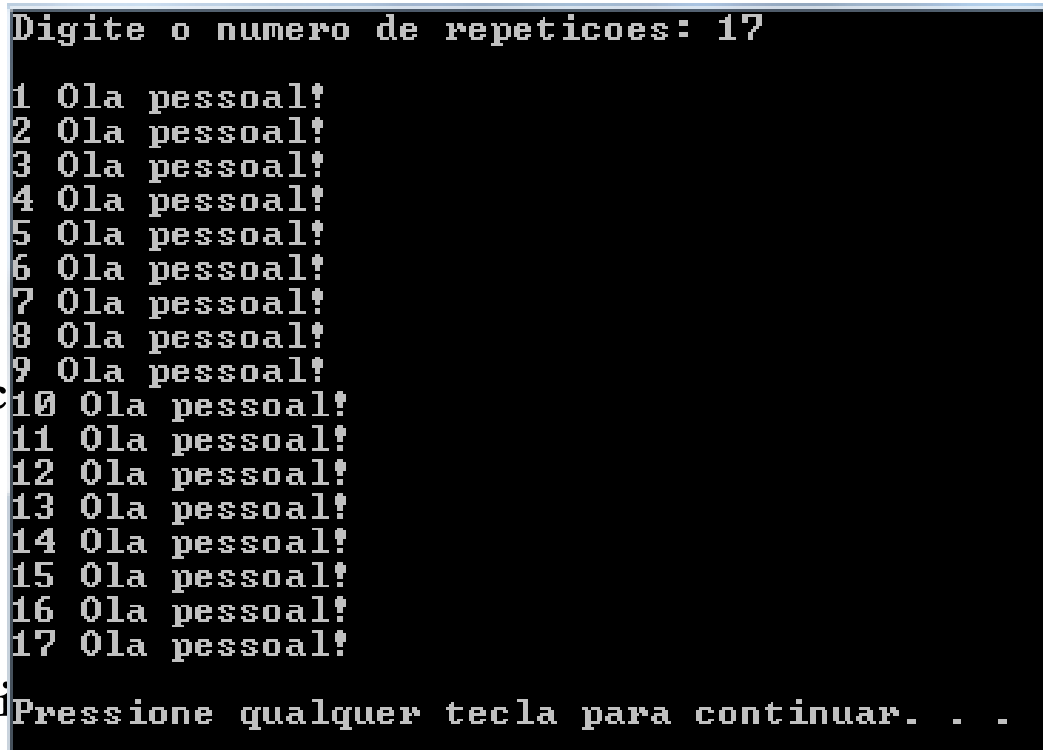
Copiar, salvar e executar este programa

Digitar um valor ≥ 10 para 'n'

Programa 2.1: Numeração das linhas

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

int main () {
    int i, n;
    printf ("Digite o numero de repetic
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%d Ola pessoal!", i
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```



Quando n ≥ 10, tem-se um desalinhamento

Alterar o programa para:

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%4d Ola pessoal!", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

%4d: escreve o valor de 'i' num espaço de 4 casas na tela

O valor de 'i' fica justaposto à direita

Alterar o programa para:

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

int main () {
    int i, n;
    printf ("Digite o numero de repe
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%4d Ola pessoa
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

%4d: escreve o valor de 'i' num espaço de 4 casas na tela

O valor de 'i' fica justaposto à direita

```
Digite o numero de repeticoes: 17

   1 Ola pessoal!
   2 Ola pessoal!
   3 Ola pessoal!
   4 Ola pessoal!
   5 Ola pessoal!
   6 Ola pessoal!
   7 Ola pessoal!
   8 Ola pessoal!
   9 Ola pessoal!
  10 Ola pessoal!
  11 Ola pessoal!
  12 Ola pessoal!
  13 Ola pessoal!
  14 Ola pessoal!
  15 Ola pessoal!
  16 Ola pessoal!
  17 Ola pessoal!

Pressione qualquer tecla para continuar. . .
```

Alterar o programa para:

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%4d Ola pessoal!", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

%4d: escreve o valor de 'i' num espaço de 4 casas na tela

O valor de 'i' fica justaposto à direita

Exemplo: se **i** 43

Na tela:

```
43_
```

4 casas

Alterar o programa para:

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%4d Ola pessoal!", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

%-4d

%4d: escreve o valor de 'i' num espaço de 4 casas na tela

O valor de 'i' fica justaposto à direita

Exemplo: se i = 43

Na tela:

```
  43_
```

4 casas

Se fosse %-4d: justaposição à esquerda:

Na tela:

```
43  _
```

4 casas

Alterar o programa para:

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

int main () {
    int i, n;
    printf ("Digite o numero de repe
    scanf ("%d", &n);
    i = 1;
    while (i <= n) {
        printf ("\n%4d Ola pessoal! ", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause");
    return 0;
}
```

```
Digite o numero de repeticoes: 17

1    Ola pessoal!
2    Ola pessoal!
3    Ola pessoal!
4    Ola pessoal!
5    Ola pessoal!
6    Ola pessoal!
7    Ola pessoal!
8    Ola pessoal!
9    Ola pessoal!
10   Ola pessoal!
11   Ola pessoal!
12   Ola pessoal!
13   Ola pessoal!
14   Ola pessoal!
15   Ola pessoal!
16   Ola pessoal!
17   Ola pessoal!

Pressione qualquer tecla para continuar. . .
```

%-4d

Se fosse %-4d: justaposição à esquerda:

Na tela: 43 _

4 casas

Alterar o programa e executar:

Deixa a saída com formato de tabela

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    printf ("\n   i  mensagem");
    printf ("\n--------------------");
    i = 1;
    while (i <= n) {
            printf ("\n%4d Ola pessoal!", i);
            i = i + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```

Alterar o programa e executar:

Deixa a saída com formato de tabela

A tabela tem um cabeçalho bem simples

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    printf ("\n   i  mensagem");
    printf ("\n--------------------");
    i = 1;
    while (i <= n) {
        printf ("\n%4d Ola pessoal!",
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```

```
Digite o numero de repeticoes: 17

   i  mensagem
--------------------
   1 Ola pessoal!
   2 Ola pessoal!
   3 Ola pessoal!
   4 Ola pessoal!
   5 Ola pessoal!
   6 Ola pessoal!
   7 Ola pessoal!
   8 Ola pessoal!
   9 Ola pessoal!
  10 Ola pessoal!
  11 Ola pessoal!
  12 Ola pessoal!
  13 Ola pessoal!
  14 Ola pessoal!
  15 Ola pessoal!
  16 Ola pessoal!
  17 Ola pessoal!

Pressione qualquer tecla para continuar. . .
```

Alterar o programa e executar:

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    printf ("\n   i |  mensagem");
    printf ("\n-----|----------------");
    i = 1;
    while (i <= n) {
        printf ("\n%4d | Ola pessoal", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```

Usa-se a barra vertical '|' para demarcar a separação das colunas

A colocação de '|' deve ser coerente nos 3 printf's

Alterar o programa e executar:

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

int main () {
    int i, n;
    printf ("Digite o numero de repeticoes: ");
    scanf ("%d", &n);
    printf ("\n   i |  mensagem");
    printf ("\n-----|----------------");
    i = 1;
    while (i <= n) {
        printf ("\n%4d | Ola pessoal", i);
        i = i + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```

Usa-se a barra vertical '|' para demarcar a separação das colunas

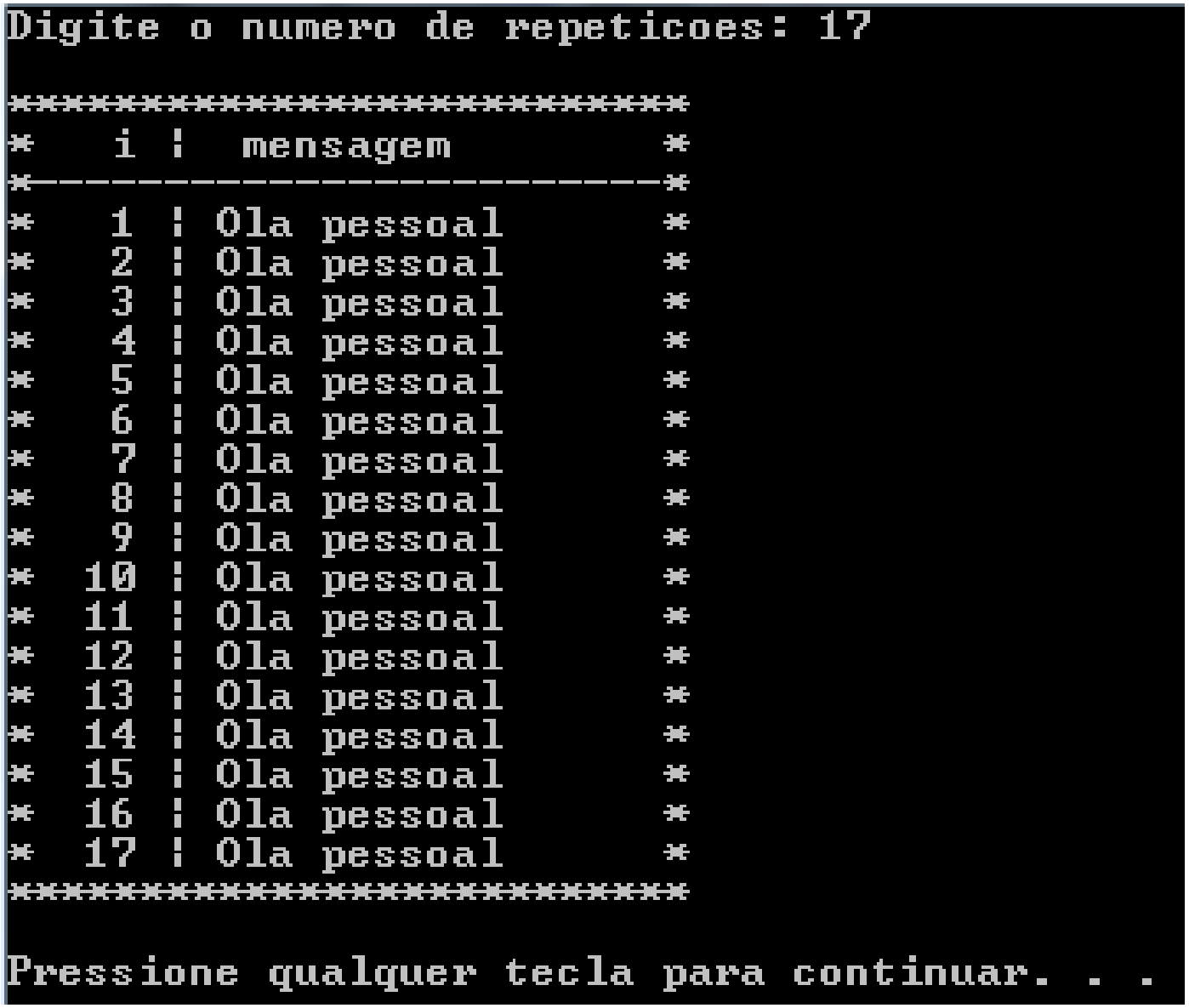
A colocação de '|' deve ser coerente nos 3 printf's

```
Digite o numero de repeticoes: 17

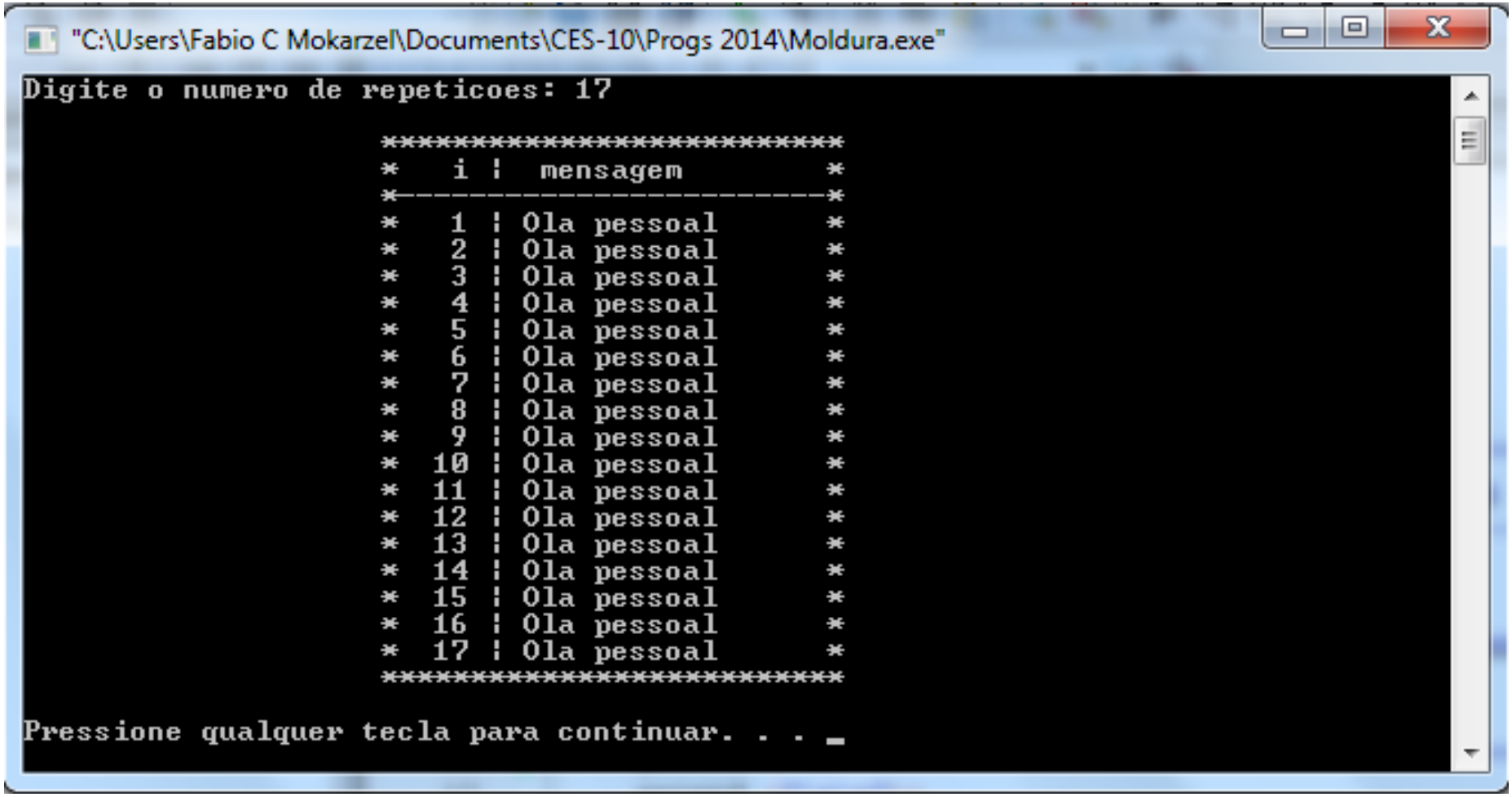
   i |  mensagem
-----|----------------
   1 | Ola pessoal
   2 | Ola pessoal
   3 | Ola pessoal
   4 | Ola pessoal
   5 | Ola pessoal
   6 | Ola pessoal
   7 | Ola pessoal
   8 | Ola pessoal
   9 | Ola pessoal
  10 | Ola pessoal
  11 | Ola pessoal
  12 | Ola pessoal
  13 | Ola pessoal
  14 | Ola pessoal
  15 | Ola pessoal
  16 | Ola pessoal
  17 | Ola pessoal

Pressione qualquer tecla para continuar. . .
```

Exercício 2.1: Colocar uma moldura na tabela, como esta:



Exercício 2.2: Deslocar a tabela mais para o meio da tela, tal como:



Programa 2.2: Tabela para x versus x^2 (x – inteiro)

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

int main () {
    int x, n;
    printf ("Digite o limite superior da tabela: ");
    scanf ("%d", &n);
    printf ("\n   x       x^2");
    printf ("\n--------------------");
    x = 1;
    while (x <= n) {
          printf ("\n%4d%10d", x, x*x);
          x = x + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```

Programa 2.2: Tabela para x versus x^2 (x – inteiro)

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

int main () {
    int x, n;
    printf ("Digite o limite superior da tabela: ");
    scanf ("%d", &n);
    printf ("\n   x      x^2");
    printf ("\n--------------------");
    x = 1;
    while (x <= n) {
        printf ("\n%4d%10d", x, x*x);
        x = x + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```

O limite inferior é 1

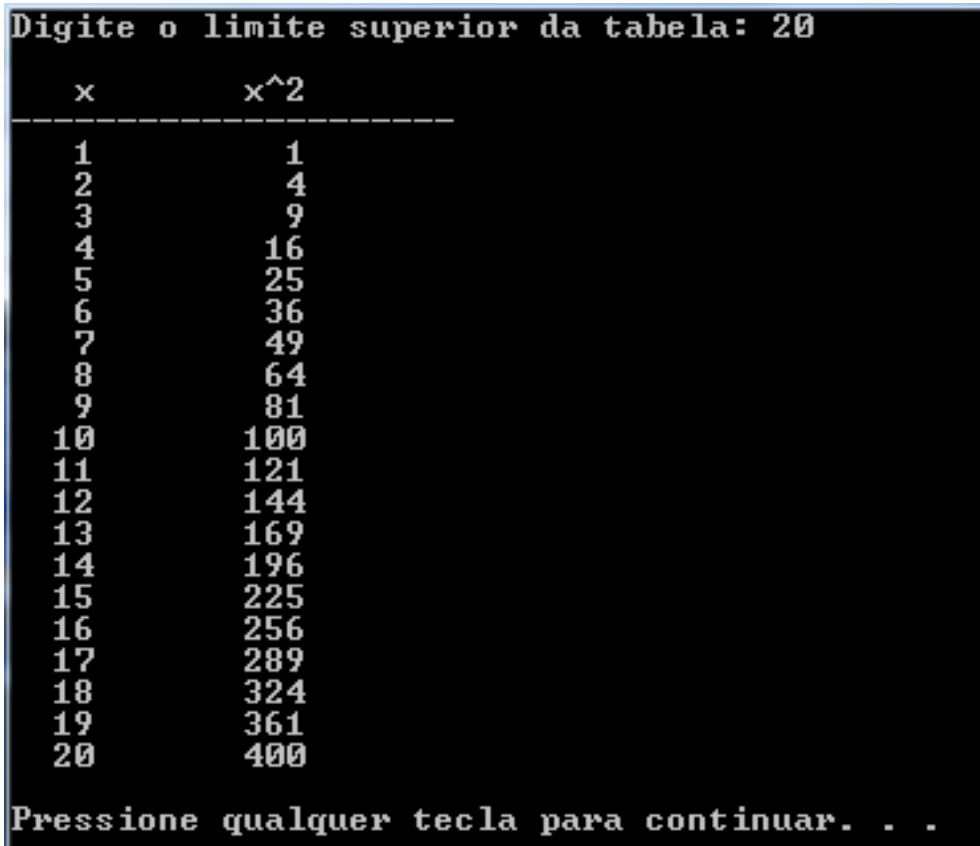
O limite superior é 'n' (digitado)

Em cada linha, um valor de x e esse valor ao quadrado

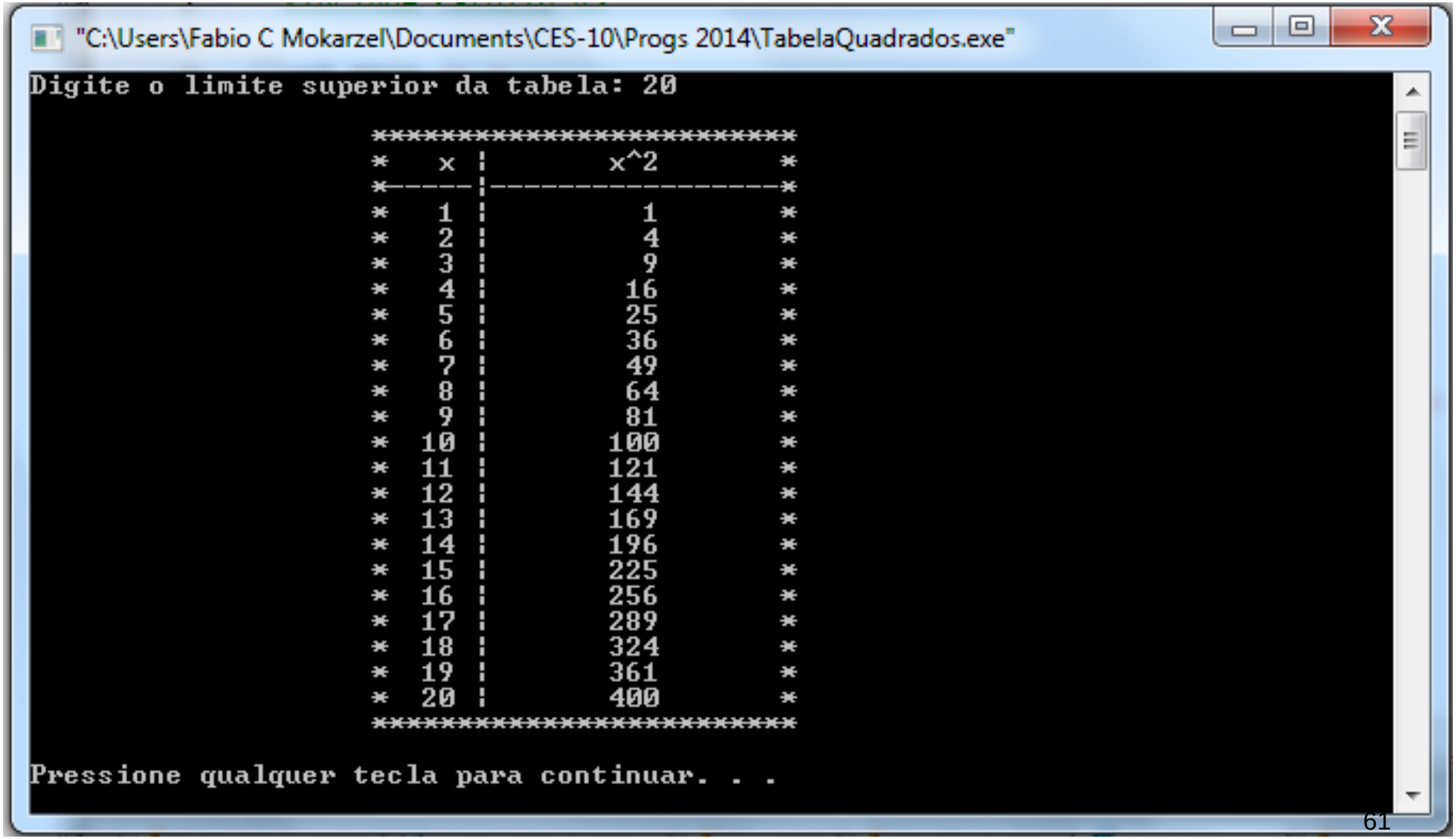
Programa 2.2: Tabela para x versus x^2 (x – inteiro)

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

int main () {
    int x, n;
    printf ("Digite o limite superior da tabela: ");
    scanf ("%d", &n);
    printf ("\n   x       x^2");
    printf ("\n--------------------");
    x = 1;
    while (x <= n) {
        printf ("\n%4d%10d", x, x*x);
        x = x + 1;
    }
    printf ("\n\n");
    system ("pause"); return 0;
}
```



Exercício 2.3: Demarcar a separação das colunas, fazer uma moldura e deslocar a tabela mais para o meio da tela, tal como:



Programa 2.3: Tabela para x (int) versus sqrt(x) (real)

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

int main () {
    int x, n;
    printf ("Digite o limite superior da tabela: ");
    scanf ("%d", &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n--------------------");
    x = 1;
    while (x <= n) {
        printf ("\n%4d%15.8f", x, sqrt(x));
        x = x + 1;
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Contém a função sqrt

%15.8f: escreve o valor real de sqrt(x) num espaço de 15 casas na tela, com 8 casas após o ponto decimal

Programa 2.3: Tabela para x (int) versus sqrt(x) (real)

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

int main () {
    int x, n;
    printf ("Digite o limite superior da tab
    scanf ("%d", &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n--------------------");
    x = 1;
    while (x <= n) {
        printf ("\n%4d%15.8f", x, sqrt(x
        x = x + 1;
    }
    printf ("\n\n"); system ("pause"); re
}
```

```
Digite o limite superior da tabela: 20

   x         sqrt(x)
--------------------
   1      1.00000000
   2      1.41421356
   3      1.73205081
   4      2.00000000
   5      2.23606798
   6      2.44948974
   7      2.64575131
   8      2.82842712
   9      3.00000000
  10      3.16227766
  11      3.31662479
  12      3.46410162
  13      3.60555128
  14      3.74165739
  15      3.87298335
  16      4.00000000
  17      4.12310563
  18      4.24264069
  19      4.35889894
  20      4.47213595

Pressione qualquer tecla para continuar. . .
```

Alterar para ler também o limite inferior, e executar:

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

int main () {
    int x, m, n;
    printf ("Digite os limites inferior e superior da tabela: ");
    scanf ("%d%d", &m, &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n--------------------");
    x = m;
    while (x <= n) {
          printf ("\n%4d%15.8f", x, sqrt(x));
          x = x + 1;
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Alterar para ler também o limite inferior, e executar:

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

int main () {
    int x, m, n;
    printf ("Digite os limites
    scanf ("%d%d", &m, &n
    printf ("\n   x       sqrt(x)
    printf ("\n----------------
    x = m;
    while (x <= n) {
          printf ("\n%4d%15.
          x = x + 1;
    }
    printf ("\n\n"); system (
}
```

```
Digite os limites inferior e superior da tabela: 8 27

   x         sqrt(x)
--------------------
   8      2.82842712
   9      3.00000000
  10      3.16227766
  11      3.31662479
  12      3.46410162
  13      3.60555128
  14      3.74165739
  15      3.87298335
  16      4.00000000
  17      4.12310563
  18      4.24264069
  19      4.35889894
  20      4.47213595
  21      4.58257569
  22      4.69041576
  23      4.79583152
  24      4.89897949
  25      5.00000000
  26      5.09901951
  27      5.19615242

Pressione qualquer tecla para continuar. . . _
```

Alterar para ler também o limite inferior, e executar:

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

int main () {
    int x, m, n;
    printf ("Digite os limites inferior e superior da tabela: ");
    scanf ("%d%d", &m, &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n--------------------");
    x = m;
    while (x <= n) {
        printf ("\n%4d%15.8f", x, sqrt(x));
        x = x + 1;
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Haverá problema para x < 0

Alterar para ler também o limite inferior, e executar:

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

int main () {
    int x, m, n;
    printf ("Digite os limites inferior e superior da tabela: ");
    scanf ("%d%d", &m, &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n----------------
    x = m;
    while (x <= n) {
        printf ("\n%4d%1
        x = x + 1;
    }
    printf ("\n\n"); system
}
```

Haverá problema para x < 0

```
Digite os limites inferior e superior da tabela: -3 7

   x         sqrt(x)
----------------------
  -3     -1.#IND0000
  -2     -1.#IND0000
  -1     -1.#IND0000
   0      0.00000000
   1      1.00000000
   2      1.41421356
   3      1.73205081
   4      2.00000000
   5      2.23606798
   6      2.44948974
   7      2.64575131

Pressione qualquer tecla para continuar. . .
```

Uma solução: usar comando **if**:

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

int main () {
    int x, m, n;
    printf ("Digite os limites inferior e superior da tabela: ");
    scanf ("%d%d", &m, &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n--------------------");
    x = m;
    while (x <= n) {
        if (x >= 0)
            printf ("\n%4d%15.8f", x, sqrt(x));
        else
            printf ("\n%4d%15s", x, "----------");
        x = x + 1;
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Comando condicional if-else:

Se x >= 0, executa o primeiro printf
Senão, executa o segundo printf

%15s: formato para escrever uma cadeia de caracteres

Escreve ---------- num espaço de 15 casas justaposto à direita

Uma solução: usar comando **if**:

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

int main () {
    int x, m, n;
    printf ("Digite os limites inferior e superior da tabela: ");
    scanf ("%d%d", &m, &n);
    printf ("\n   x       sqrt(x)");
    printf ("\n--------------------");
    x = m;
    while (x <= n) {
        if (x >= 0)
            printf ("\n%
        else
            printf ("\n%
        x = x + 1;
    }
    printf ("\n\n"); system ("p
}
```

```
Digite os limites inferior e superior da tabela: -3 7

   x         sqrt(x)
--------------------
  -3      ----------
  -2      ----------
  -1      ----------
   0      0.00000000
   1      1.00000000
   2      1.41421356
   3      1.73205081
   4      2.00000000
   5      2.23606798
   6      2.44948974
   7      2.64575131

Pressione qualquer tecla para continuar. . . _
```

2º Exercício Laboratorial: Fazer um programa para exibir uma única tabela com as seguintes funções:

- Função 1: $\mathbf{x^4 - 20x^2 + 5}$
- Função 2: $\mathbf{\sin(x*\pi/180)}$
- Função 3: $\mathbf{x^7/\log_e x}$

} **No intervalo [m, n]**

- Os limites **m** e **n** do intervalo são inteiros e devem ser digitados pelo operador

- A tabela deve ter demarcação da separação das **colunas**, **moldura** e **deslocamento** mais para o meio da tela

Função sin(x), para x dado em graus:

- O arquivo **math.h** tem a função **sin(x)**, para **x** dado em **radianos**

- Então, para cada valor de **x** em graus, a chamada da função deve ser:

sin(x*π/180), ou seja, **sin(x*3.14159/180)**

- A precisão de **π** fica a critério do programador

Função $x^7/\log_e x$:

- O arquivo **math.h** tem as seguintes funções:

  pow (x, y), para calcular $\mathbf{x^y}$

  log(x), para calcular $\mathbf{\log_e x}$

- Para $\mathbf{x \leq 0}$ e $\mathbf{x = 1}$, a função tem problemas

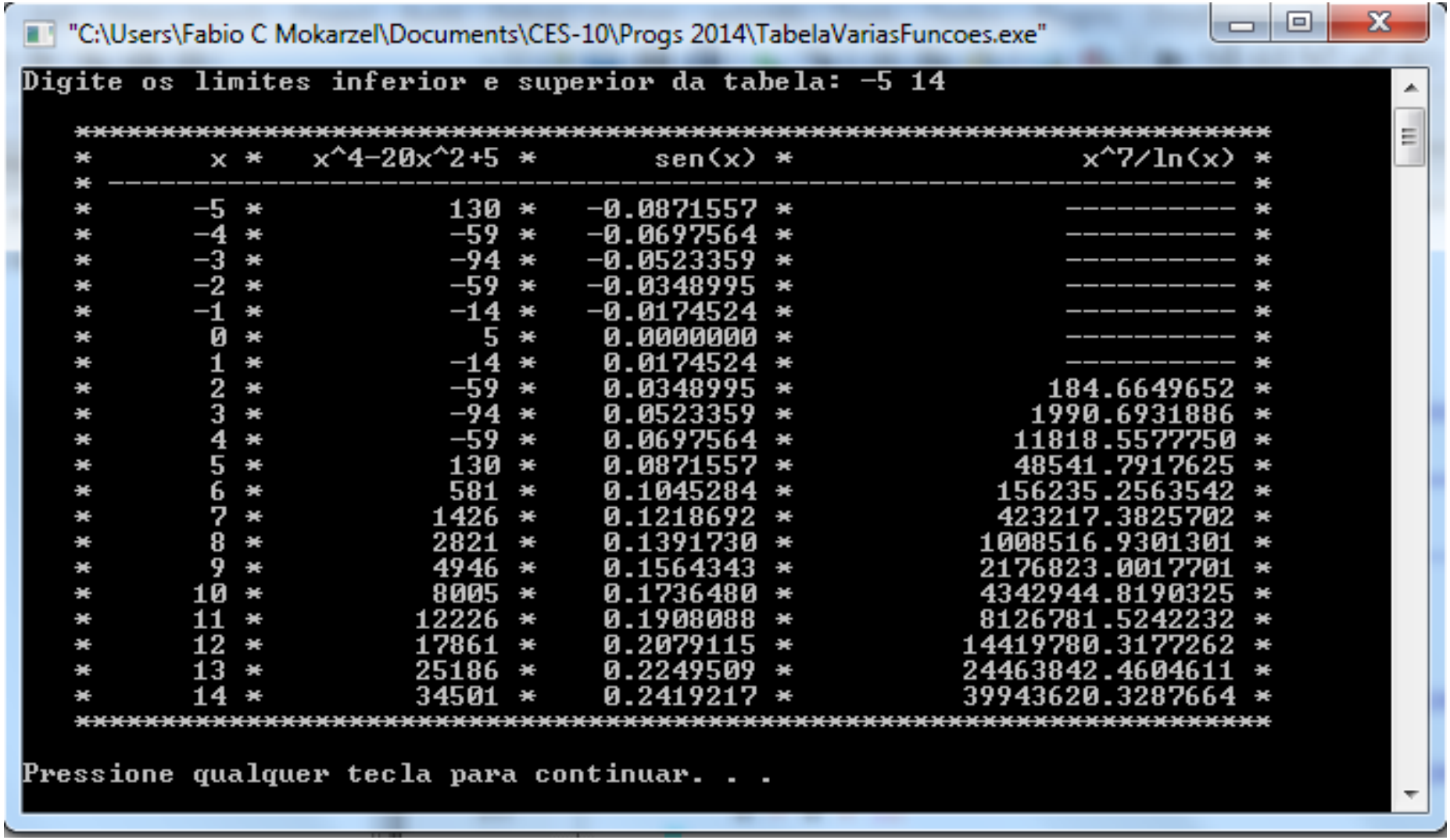
Função $x^4 - 20x^2 + 5$:

- Usar a seguinte programação:

x*x*x*x - 20*x*x + 5

Para evitar resultados reais fornecidos pela função **pow**

Exemplo para as três funções pedidas:



Programa 2.4: Variáveis e valores do tipo caractere

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char c;
    printf ("Digite uma tecla: ");
    scanf ("%c", &c);
    printf ("decimal = %d, caractere = %c", c, c);
    printf ("\n\n"); system ("pause");
    return 0;
}
```

Variável do tipo caractere: destina-se a guardar um caractere num dado momento

- **Caractere** é um símbolo de grafia

- **Exemplos de caracteres:**

  - Letras maiúsculas e minúsculas: **A, B, ... , Z, a, b, ... , z**
  - Dígitos decimais: **0, 1, 2, ... , 8, 9**
  - Operadores aritméticos: **+, -, *, /**
  - Separadores: **(,), [,], {, }**
  - Caracteres de pontuação: **. , ; ! ? :**
  - Caracteres especiais: **@, #, $, %, &**
  - Caracteres de controle: muda linha **(nl)**, tabulação **(ht),** etc.

Cada caractere tem sua **representação interna** segundo a **tabela ASCII**:

Livro 02

	0	1	2	3	4	5	6	7	8	9
0	nul							bel	bs	ht
10	nl			cr						
20								esc		
30			sp	!	“	#	$	%	&	‘
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	del		

Exemplos:

ASCII (‘K’) = 75

ASCII (‘&’) = 38

ASCII (‘{’) = 123

Esses valores são as representações internas dos caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char c;
    printf ("Digite uma tecla: ");
    scanf ("%c", &c);
    printf ("decimal = %d, caractere = %c", c, c);
    printf ("\n\n"); system ("pause");
    return 0;
}
```

scanf ("%c", &c): espera o operador digitar uma tecla seguida de <enter>

Em seguida, a variável 'c' recebe a representação interna do caractere correspondente a essa tecla

Algumas execuções:

```
Digite uma tecla: B
decimal = 66, caractere = B

Pressione qualquer tecla para cor
```

```
Digite uma tecla: #
decimal = 35, caractere = #

Pressione qualquer tecla para cor
```

```
Digite uma tecla: x
decimal = 120, caractere = x

Pressione qualquer tecla para cor
```

```
Digite uma tecla: 7
decimal = 55, caractere = 7

Pressione qualquer tecla para cor
```

```
Digite uma tecla: >
decimal = 62, caractere = >

Pressione qualquer tecla para cor
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char c;
    printf ("Digite uma tecla: ");
    scanf ("%c", &c);
    printf ("decimal = %d, caractere = %c", c, c);
    printf ("\n\n"); system ("pause");
    return 0;
}
```

%d: escreve a representação interna do caractere guardado na variável 'c'

%c: escreve o próprio caractere guardado na variável 'c'

Algumas execuções:

```
Digite uma tecla: B
decimal = 66, caractere = B

Pressione qualquer tecla para cor
```

```
Digite uma tecla: #
decimal = 35, caractere = #

Pressione qualquer tecla para cor
```

```
Digite uma tecla: x
decimal = 120, caractere = x

Pressione qualquer tecla para cor
```

```
Digite uma tecla: 7
decimal = 55, caractere = 7

Pressione qualquer tecla para cor
```

```
Digite uma tecla: >
decimal = 62, caractere = >

Pressione qualquer tecla para cor
```

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo III

Comandos de Controle

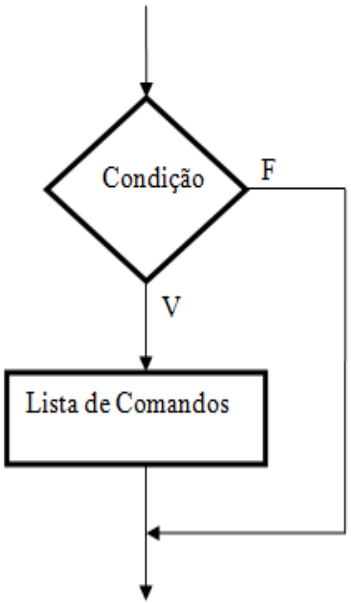
Programa 3.1: Comandos condicionais

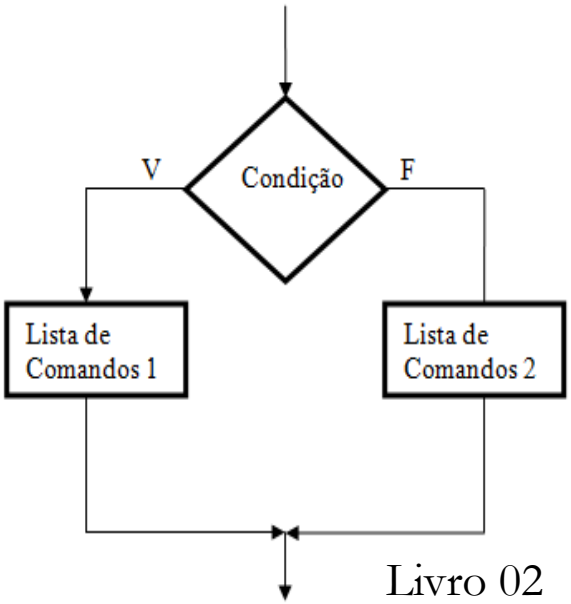
Digitar, salvar e executar o programa à esquerda

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

int main () {
    char estahfrio;
    printf ("Estah frio? (s/n): ");
    scanf ("%c", &estahfrio);
    if (estahfrio == 's' || estahfrio == 'S')
        printf ("\n\tVista seu casaco!!");
    else
        printf ("\n\tDeixe seu casaco no guarda-roupas!!");
    printf ("\n\n"); system ("pause");
    return 0;
}
```

Fluxogramas explicativos:





Livro 02

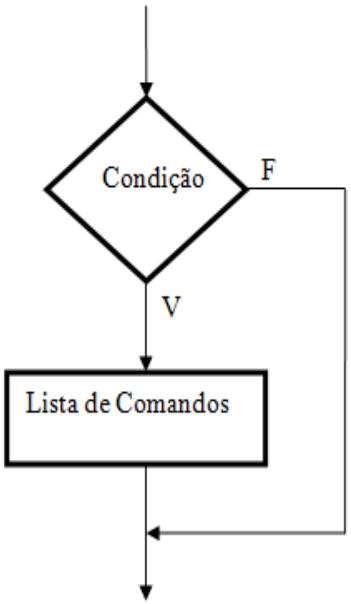
```
#include <stdio.h>
#include <stdlib.h>

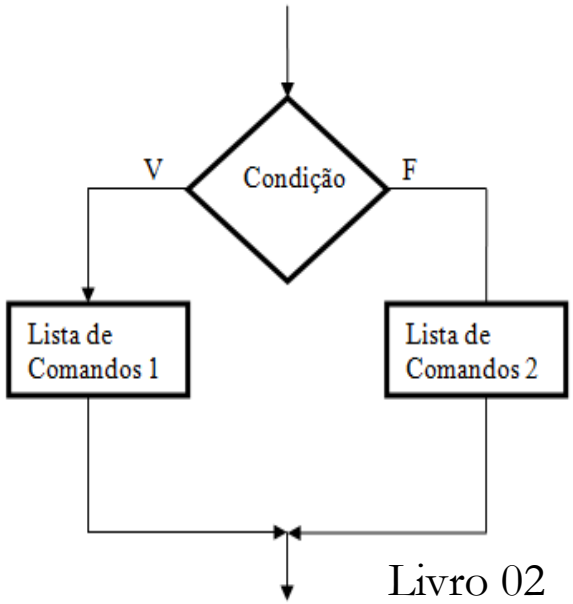
int main () {
    char estahfrio;
    printf ("Estah frio? (s/n): ");
    scanf ("%c", &estahfrio);
    if (estahfrio == 's' || estahfrio == 'S')
        printf ("\n\tVista seu casaco!!");
    else
        printf ("\n\tDeixe seu casaco no guarda-roupas!!");
    printf ("\n\n"); system ("pause");
    return 0;
}
```

==: usado para comparar dois valores

=: usado para atribuir uma expressão a uma variável

Fluxogramas explicativos:





Livro 02

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

int main () {
    char estahfrio;
    printf ("Estah frio? (s/n): ");
    scanf ("%c", &estahfrio);
    if (estahfrio == 's' || estahfrio == 'S')
        printf ("\n\tVista seu casaco!!");
    else
        printf ("\n\tDeixe seu casaco no guarda-roupas!!");
    printf ("\n\n"); system ("pause");
    return 0;
}
```

||: operador **'or'** entre duas condições

&&: operador **'and'** entre duas condições

Programa 3.2: Outro comando repetitivo: do-while

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int n; char c, lixo;
    printf ("PAR OU IMPAR");
    do {
        printf ("\n\nDigite um numero seguido de <enter>: ");
        scanf ("%d%c", &n, &lixo);
        if (n % 2 == 0)
            printf ("\t%d eh par", n);
        else
            printf ("\t%d eh impar", n);
        printf ("\n\nVerificar paridade? (s/n): ");
        scanf ("%c%c", &c, &lixo);
    } while (c == 's' || c == 'S');
    printf ("\n\n"); system ("pause"); return 0;
}
```

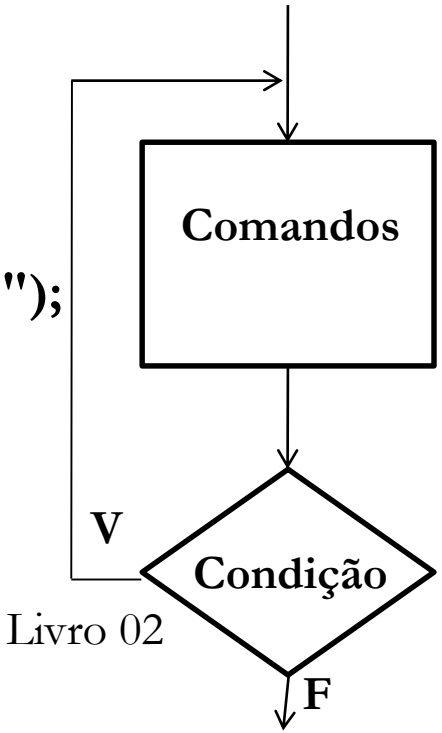
Digitar, salvar e executar o programa á esquerda

Programa 3.2: Outro comando repetitivo: do-while

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int n; char c, lixo;
    printf ("PAR OU IMPAR");
    do {
        printf ("\n\nDigite um numero seguido de <enter>: ");
        scanf ("%d%c", &n, &lixo);
        if (n % 2 == 0)
            printf ("\t%d eh par", n);
        else
            printf ("\t%d eh impar", n);
        printf ("\n\nVerificar paridade? (s/n): ");
        scanf ("%c%c", &c, &lixo);
    } while (c == 's' || c == 'S');
    printf ("\n\n"); system ("pause"); return 0;
}
```

n % 2: resto da divisão de n por 2

Fluxograma explicativo



Livro 02

Os comandos são executados pelo menos uma vez

Programa 3.2: Outro comando repetitivo: do-while

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int n; char c, lixo;
    printf ("PAR OU IMPAR");
    do {
        printf ("\n\nDigite um numero seguido de <enter>: ");
        scanf ("%d%c", &n, &lixo);
        if (n % 2 == 0)
                printf ("\t%d eh par", n);
        else
                printf ("\t%d eh impar", n);
        printf ("\n\nVerificar paridade? (s/n): ");
        scanf ("%c%c", &c, &lixo);
    } while (c == 's' || c == 'S');
    printf ("\n\n"); system ("pause"); return 0;
}
```

Para o scanf, tudo o que é digitado, inclusive o <enter>, fica depositado numa região chamada "buffer do teclado"

O computador só espera digitação se esse buffer estiver vazio

Depois da leitura para n e c, o <enter> fica no buffer e será usado na próxima leitura

Nos dois scanf's, o <enter> será lido e guardado na variável lixo

Programa 3.2: Outro comando repetitivo: do-while

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int n; char c, lixo;
    printf ("PAR OU IMPAR");
    do {
        printf ("\n\nDigite um numero seguido de <enter>: ");
        scanf ("%d%c", &n, &lixo);
        if (n % 2 == 0)
                printf ("\t%d eh par", n);
        else
                printf ("\t%d eh impar", n);
        printf ("\n\nVerificar paridade? (s/n): ");
        scanf ("%c%c", &c, &lixo);
    } while (c == 's' || c == 'S');
    printf ("\n\n"); system ("pause"); return 0;
}
```

Para o scanf, tudo o que é digitado, inclusive o <enter>, fica depositado numa região chamada "buffer do teclado"

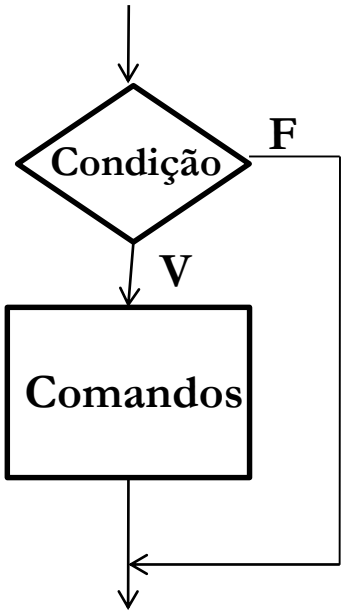
O computador só espera digitação se esse buffer estiver vazio

Se a variável lixo não fosse usada, o <enter> do primeiro scanf seria lido para a variável c, no segundo scanf

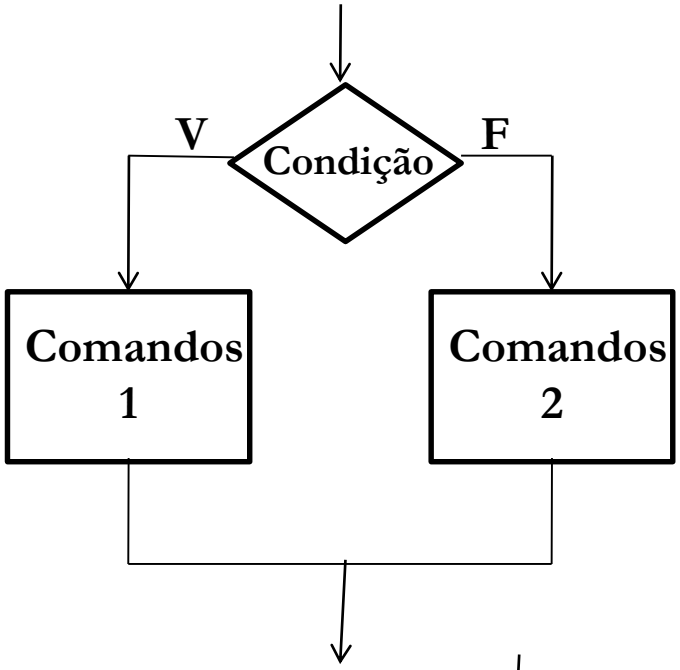
Aí o do-while se encerraria sem nova digitação para a variável c

Com as seguintes construções, escreve-se grande variedade de programas:

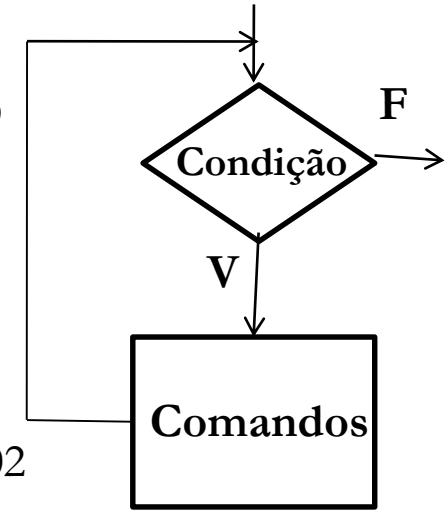
```
if (Condição)
    Comandos
```



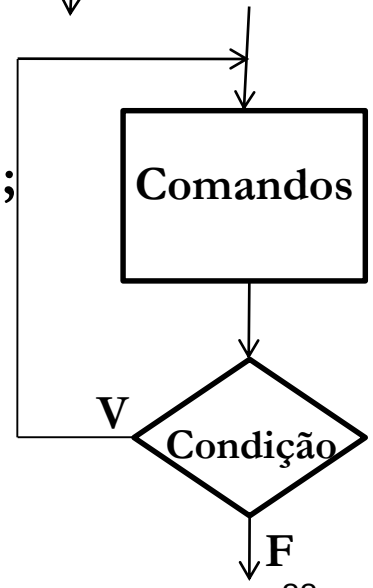

```
if (Condição)
    Comandos 1
else
    Comandos 2
```




```
while (Condição)
    Comandos
```




```
do
    Comandos
while (Condição);
```



Além dos comandos:

```
variável = expressão;
scanf (--------);
printf (--------);
```

Programa 3.3: Divisibilidade por 7 – vários números

- Em **C**, uma **divisão** em que os 2 operandos são **inteiros** produz resultado **inteiro truncado**
- **Exemplo: 7 / 3 = 2**
- Se um dos operandos for **real**, o resultado será **real**
- **Operador %: resto de divisão**
  - Exemplo: **7 % 3 = 1**
  - Os **operandos** devem ser **inteiros**

Para testar se um número 'n' é divisível por 7, basta verificar se a condição

n % 7 == 0

é satisfeita

Desenvolvimento do programa

```
Divisibilidade por 7 {
    Escrever um título inicial que diga o assunto do programa
    Perguntar ao operador se deseja testar divisibilidade por 7
    Obter a resposta do operador
    Enquanto a resposta for "sim" {
        Pedir o dividendo ao operador
        Obter o dividendo do operador
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

Algoritmo do programa, escrito em Português, num certo nível de detalhes

Divisibilidade por 7 {

Escrever um título inicial que diga o assunto do programa

Perguntar ao operador se deseja testar divisibilidade por 7

Obter a resposta do operador

Enquanto a resposta for "sim" {

Pedir o dividendo ao operador

Obter o dividendo do operador

Testar a divisibilidade do dividendo por 7 e dar a resposta

Perguntar novamente ao operador se deseja testar divisibilidade por 7

Obter a resposta do operador

}

Fechar a tela de execução

}

```
int main () {
    Escrever um título inicial que diga o assunto do programa
    Perguntar ao operador se deseja testar divisibilidade por 7
    Obter a resposta do operador
    Enquanto a resposta for "sim" {
        Pedir o dividendo ao operador
        Obter o dividendo do operador
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
int main () {
    Escrever um título inicial que diga o assunto do programa
    Perguntar ao operador se deseja testar divisibilidade por 7
    Obter a resposta do operador
    Enquanto a resposta for "sim" {
        Pedir o dividendo ao operador
        Obter o dividendo do operador
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    printf ("DIVISIBILIDADE   POR   7");
    Perguntar ao operador se deseja testar divisibilidade por 7
    Obter a resposta do operador
    Enquanto a resposta for “sim” {
            Pedir o dividendo ao operador
            Obter o dividendo do operador
            Testar a divisibilidade do dividendo por 7 e dar a resposta
            Perguntar novamente ao operador se deseja testar
                    divisibilidade por 7
            Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    printf ("DIVISIBILIDADE   POR   7");
    Perguntar ao operador se deseja testar divisibilidade por 7
    Obter a resposta do operador
    Enquanto a resposta for “sim” {
            Pedir o dividendo ao operador
            Obter o dividendo do operador
            Testar a divisibilidade do dividendo por 7 e dar a resposta
            Perguntar novamente ao operador se deseja testar
                    divisibilidade por 7
            Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    char testar, lixo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    Enquanto a resposta for “sim” {
        Pedir o dividendo ao operador
        Obter o dividendo do operador
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    char testar, lixo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    Enquanto a resposta for “sim” {
            Pedir o dividendo ao operador
            Obter o dividendo do operador
            Testar a divisibilidade do dividendo por 7 e dar a resposta
            Perguntar novamente ao operador se deseja testar
                        divisibilidade por 7
            Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    char testar, lixo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        Pedir o dividendo ao operador
        Obter o dividendo do operador
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    char testar, lixo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        Pedir o dividendo ao operador
        Obter o dividendo do operador
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    char testar, lixo; int dvdo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo seguido de <enter>: ");
        scanf ("%d%c", &dvdo, &lixo);
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>

int main () {
    char testar, lixo; int dvdo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo seguido de <enter>: ");
        scanf ("%d%c", &dvdo, &lixo);
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        Perguntar novamente ao operador se deseja testar
                divisibilidade por 7
        Obter a resposta do operador
    }
    Fechar a tela de execução
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char testar, lixo; int dvdo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo seguido de <enter>: ");
        scanf ("%d%c", &dvdo, &lixo);
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        printf ("\n\nTestar divisibilidade? (s/n): ");
        scanf ("%c%c", &testar, &lixo);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char testar, lixo; int dvdo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo seguido de <enter>: ");
        scanf ("%d%c", &dvdo, &lixo);
        Testar a divisibilidade do dividendo por 7 e dar a resposta
        printf ("\n\nTestar divisibilidade? (s/n): ");
        scanf ("%c%c", &testar, &lixo);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char testar, lixo; int dvdo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo seguido de <enter>: ");
        scanf ("%d%c", &dvdo, &lixo);
        if (dvdo % 7 == 0)
                printf ("\n\t%d eh divisivel por 7", dvdo);
        else
                printf ("\n\t%d nao eh divisivel por 7", dvdo);
        printf ("\n\nTestar divisibilidade? (s/n): ");
        scanf ("%c%c", &testar, &lixo);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char testar, lixo; int dvdo;
    printf ("DIVISIBILIDADE   POR   7");
    printf ("\n\nTestar divisibilidade? (s/n): ");
    scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo seguido de <enter>: ");
        scanf ("%d%c", &dvdo, &lixo);
        if (dvdo % 7 == 0)
                printf ("\n\t%d eh divisivel por 7", dvdo);
        else
                printf ("\n\t%d nao eh divisivel por 7", dvdo);
        printf ("\n\nTestar divisibilidade? (s/n): ");
        scanf ("%c%c", &testar, &lixo);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Salvar e executar

Este programa só testa divisibilidade por 7

Exercício 3.1: Alterar o programa anterior de forma que o divisor também seja digitado pelo operador

- **Cuidado: o divisor não pode ser zero**

- Se o operador digitar **0 (zero)** para o **divisor**, o programa deve **insistir** para ele refazer a digitação, até que **não mais cometa esse erro**

Uma possível solução

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char testar, lixo; int dvdo, dsor;
    printf ("Testar divisibilidade? (s/n): "); scanf ("%c%c", &testar, &lixo);
    while (testar == 's' || testar == 'S') {
        printf ("\n\tDigite o dividendo e o divisor seguidos de <enter>: ");
        scanf ("%d%d%c", &dvdo, &dsor, &lixo);
        while (dsor == 0) {
            printf ("\n\tDigite novamente o divisor seguido de <enter>: ");
            scanf ("%d%c", &dsor, &lixo);
        }
        if (dvdo % dsor == 0)
            printf ("\n\t%d eh divisivel por %d", dvdo, dsor);
        else
            printf ("\n\t%d nao eh divisivel por %d", dvdo, dsor);
        printf ("\n\nTestar divisibilidade? (s/n): ");
        scanf ("%c%c", &testar, &lixo);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

3º Exercício Laboratorial: Escrever um programa amigável em C para:

- Inicialmente, ler um **número inteiro positivo** digitado e guardá-lo na variável **n**
  - Caso o operador não digite um número **positivo** para **n**, o programa deve **insistir** para ele refazer a digitação, até que **não mais cometa esse erro**
- Em seguida, o programa deve ler **n** números inteiros digitados

Continua na próxima página

- Realizar com os números inteiros lidos as seguintes tarefas:

1. Contabilizar e imprimir a **média aritmética** dos números lidos

    - Inicializar uma variável **soma** com **zero** e com **zero** também uma variável **cont**, contadora de números lidos

    - A cada número lido **num**, fazer

      soma = soma + num;

      cont = cont + 1;

    - No final, dividir **soma** por **cont** (resultado real)

- Realizar com os números inteiros lidos as seguintes tarefas:

2. Contabilizar e imprimir a **soma** dos números lidos que são **pares**

    - Inicializar outra variável **somapares** com **zero**

    - A cada número lido **num** que for par, fazer

somapares = somapares + num;

- Realizar com os números inteiros lidos as seguintes tarefas:

3. Contabilizar e imprimir a **soma** dos números lidos **positivos** que são **múltiplos de 3** ou **de 7**

    - Inicializar outra variável **soma3ou7** com **zero**

    - A cada número lido **num** que for positivo e múltiplo de 3 ou de 7, fazer

soma3ou7 = soma3ou7 + num;

- Realizar com os números inteiros lidos as seguintes tarefas:

4. Para cada número **positivo** lido, notificar o caso de ele ser o **fatorial** de algum número inteiro positivo

5. Encontrar e imprimir **o maior e o menor** em valores **relativos** dentre todos os números lidos

    - Inicializar duas variáveis **maior** e **menor** com o primeiro dos **n** números digitados

    - A cada novo número lido **num**, caso ele seja maior relativamente que o valor de **maior**, fazer

      maior = num;

    - Analogamente para o valor de **menor**

- Realizar com os números inteiros lidos as seguintes tarefas:

6. Para cada número **positivo** lido, notificar o caso de ele ser um dos termos da **Série de Fibonacci**

   - A Série de Fibonacci é a seguinte: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ……..,** onde

     – O primeiro termo é 0 (zero), o segundo é 1 (um)

     – Os outros termos são dados por

     termo[i] = termo [i-1] + termo [i-2]

   - Não usar as chamadas variáveis indexadas

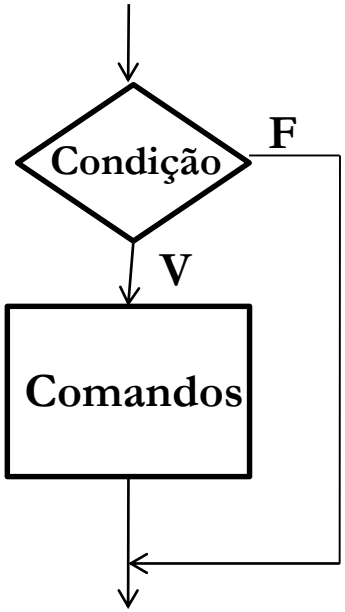
AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo IV

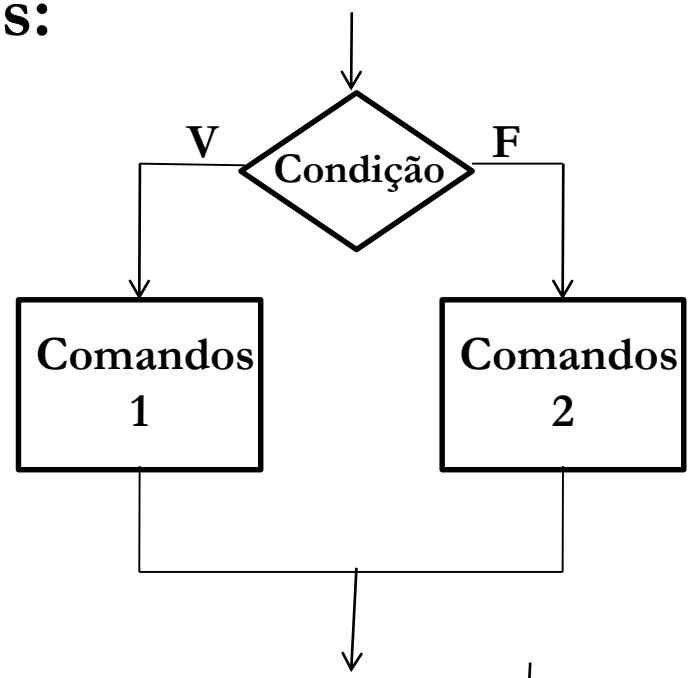
Fluxogramas

Fluxogramas foram usados até aqui para descrever comandos condicionais e repetitivos:

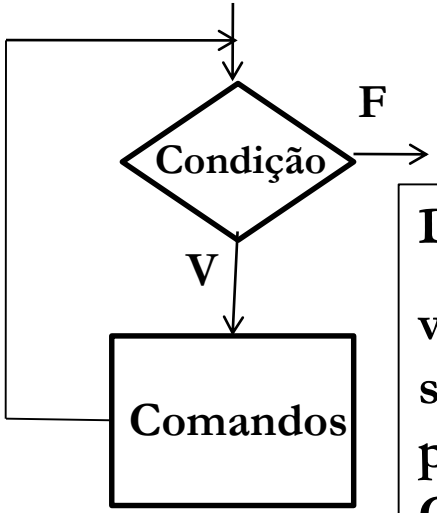
```
if (Condição)
    Comandos
```



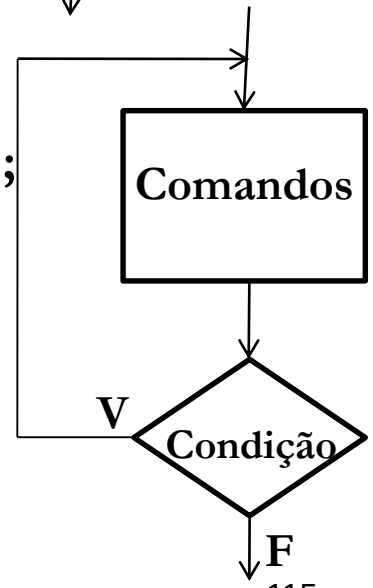

```
if (Condição)
    Comandos 1
else
    Comandos 2
```




```
while (Condição)
    Comandos
```




```
do
    Comandos
while (Condição);
```



Dentro dos comandos:

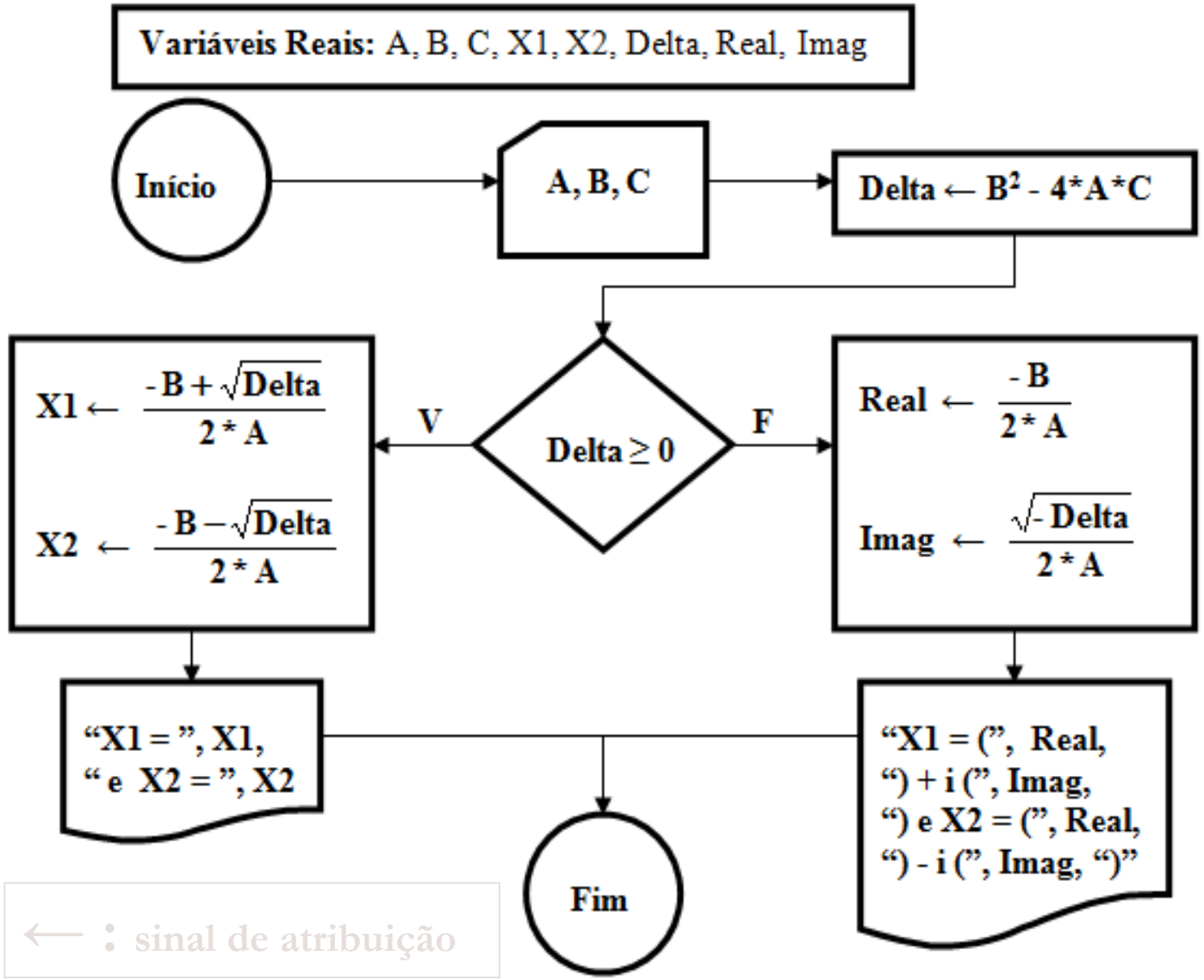
```
variável = expressão;
scanf (--------);
printf (--------);
```

Condicionais e repetitivos

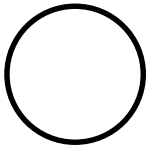
- **Fluxogramas** podem ser considerados como uma **linguagem** para descrição de **algoritmos**

- Uma vez escrito o **fluxograma**, ele então é **traduzido** para uma **linguagem de programação**

Exemplo: raízes da equação do 2º grau $Ax^2 + Bx + C = 0$



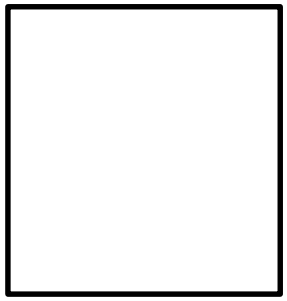
Blocos dos fluxogramas:



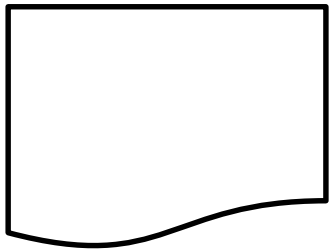
Círculo: início e final do algoritmo



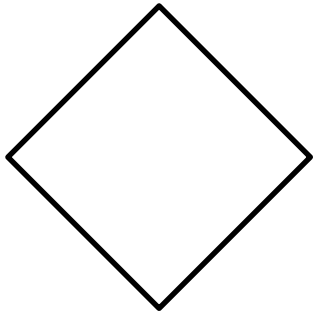
Cartão: entrada de dados



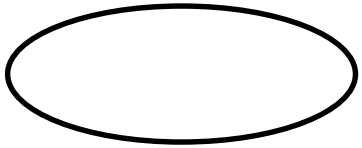
Retângulo: comandos (atribuição ou chamada de subprograma) a serem executados sequencialmente



Folha de papel: saída de resultados



Losango: decisão por um de dois caminhos alternativos

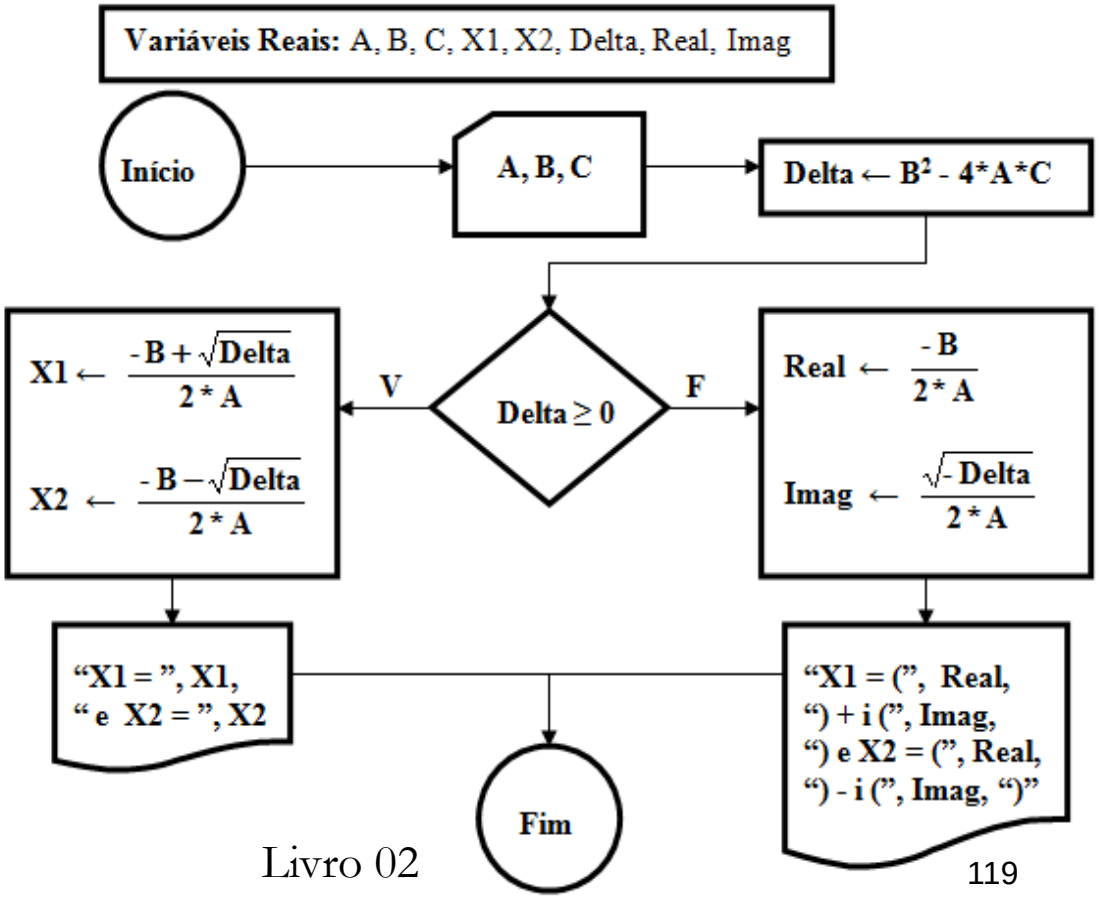


Elipse: decisão por um de vários caminhos alternativos

```
#include <stdio.h>      -   Usa scanf e printf
#include <stdlib.h>     -   Usa system para encerrar
#include <math.h>       -   B² usa pow; raiz quadrada usa sqrt
int main () {
   - - -
}
```

Tradução do fluxograma para a Linguagem C

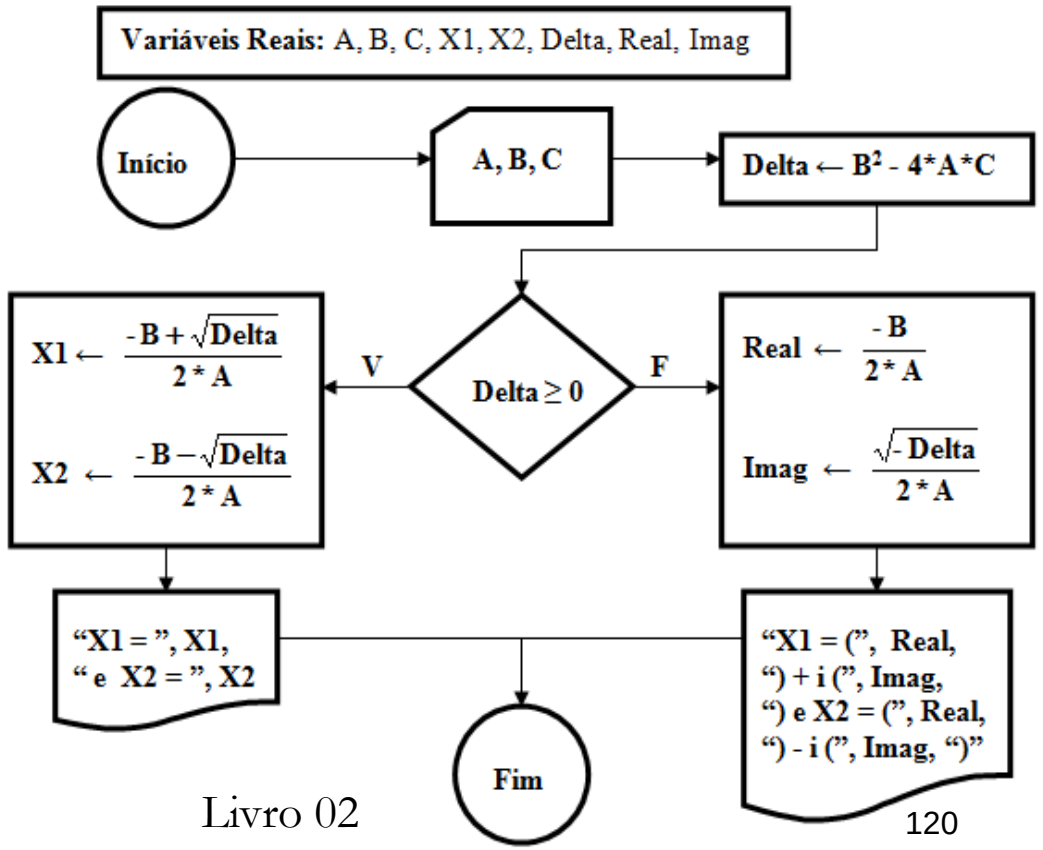
Digitar para salvar e executar posteriormente



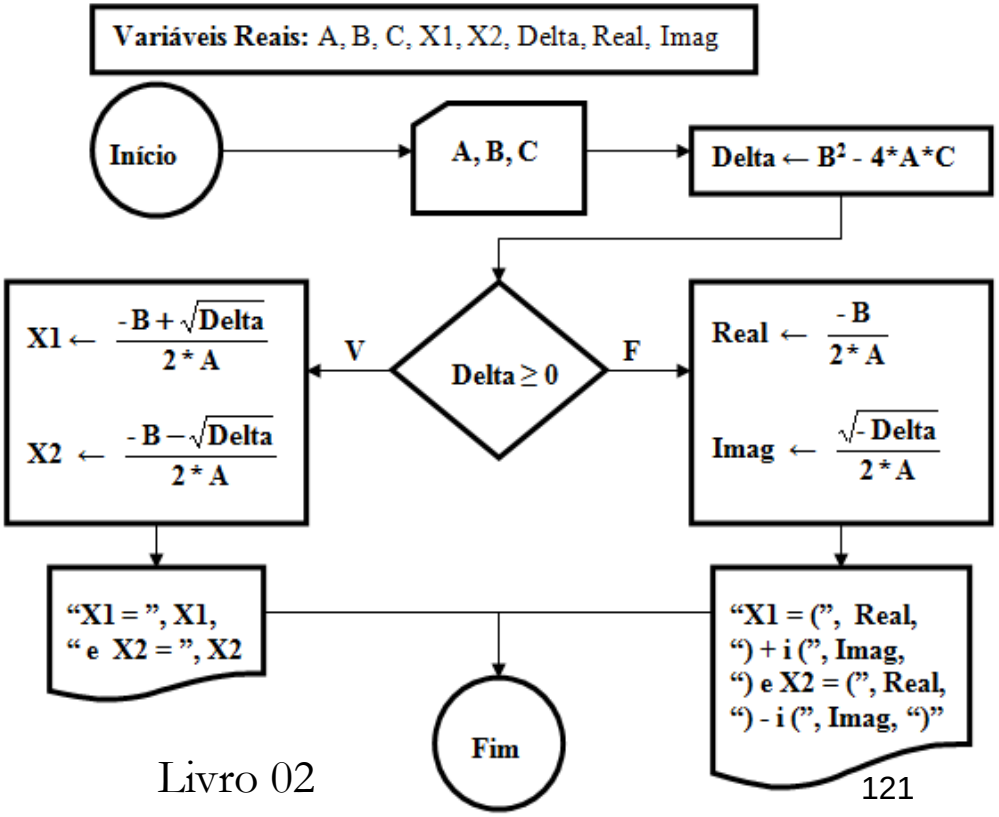
```
float A, B, C, X1, X2, Delta, Real, Imag;
printf (“Digite os coeficientes de uma equacao de 2o grau: ”);
scanf (“%f%f%f”, &A, &B, &C);
Delta = pow(B,2) – 4*A*C;
if (Delta >= 0) {
    - - - - -
}
else {
    - - - - -
}
printf  ("\n\n");
system ("pause");
return 0;
```

No escopo de main

O fluxograma não precisa estar amigável, mas o programa sim



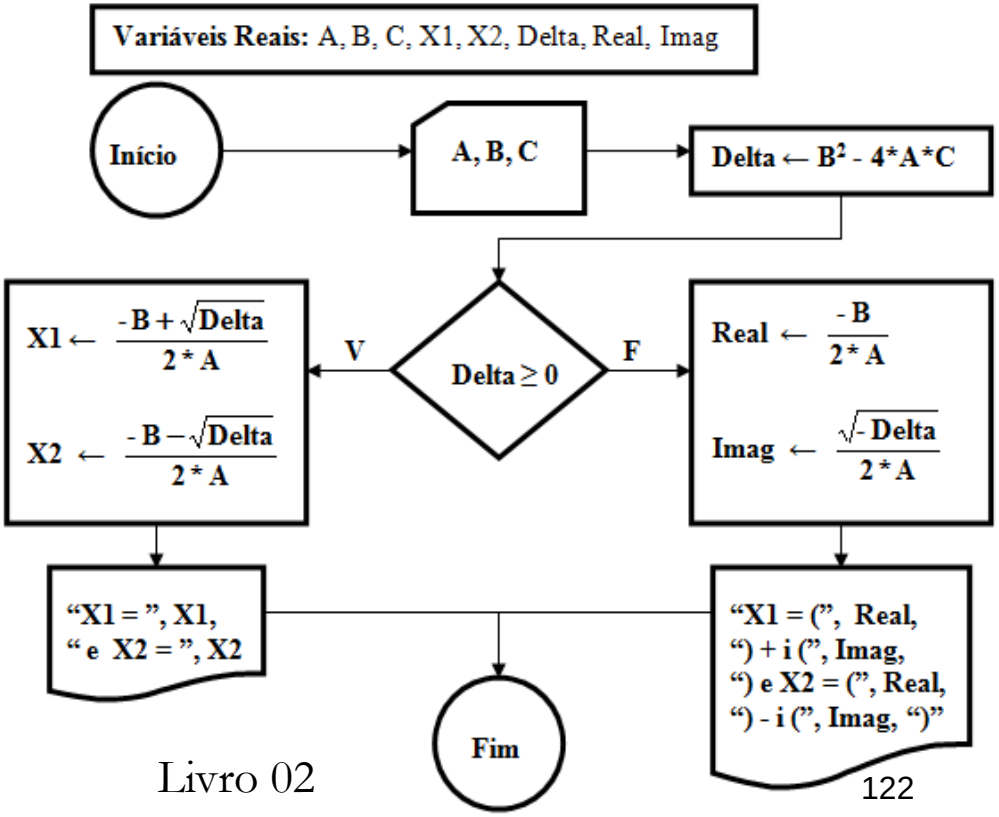
```
if (Delta >= 0) {
    X1 = (-B + sqrt(Delta)) / (2*A);
    X2 = (-B - sqrt(Delta)) / (2*A);
    write (- - - - -);
}
else {
    Real = -B / (2*A);
    Imag = sqrt(-Delta) / (2*A);
    write (- - - - -);
}
```



```
if (Delta >= 0) {- - - - -
    write ("X1 = ",  X1,  " e X2 = ",  X2);
}
else {- - - - -
    write ("X1 = (",  Real,  ")+i(",  Imag,  ") e X2 = (",  Real,  ")-i(",
          Imag,  ")");
}
```

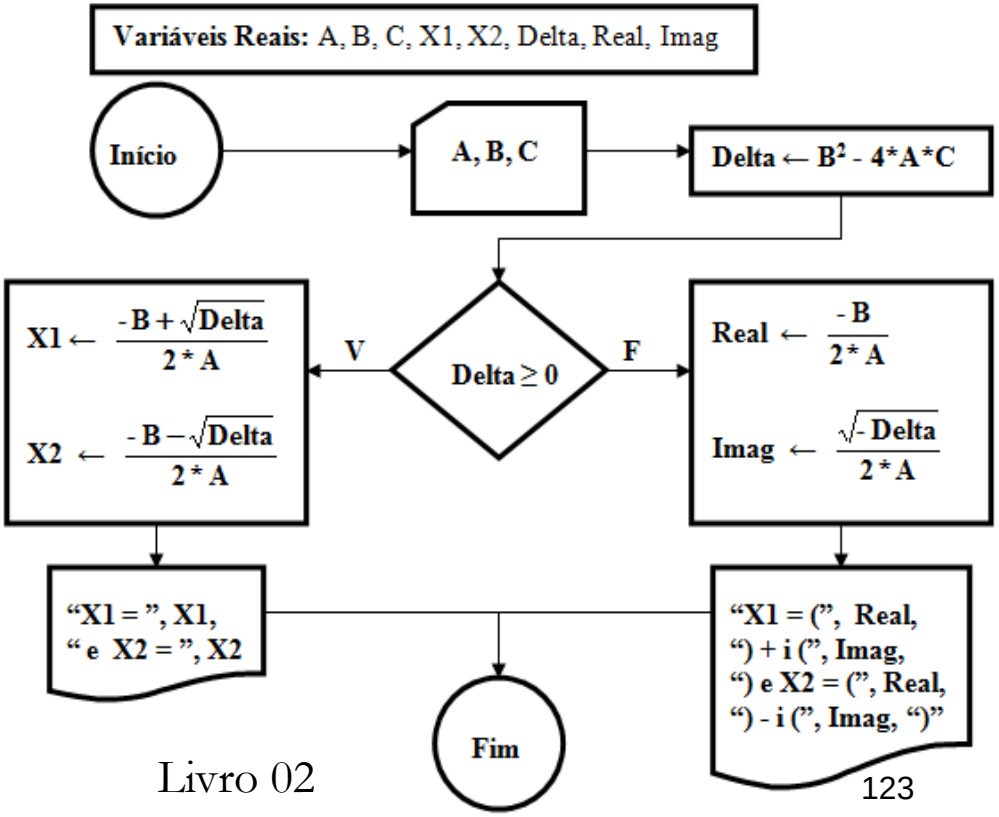
As chamadas de write contém os elementos a serem escritos

Não estão em C



```
write ("X1 = ",  X1,  " e X2 = ",  X2);
    printf ("\nX1 = %g e X2 = %g", X1, X2);

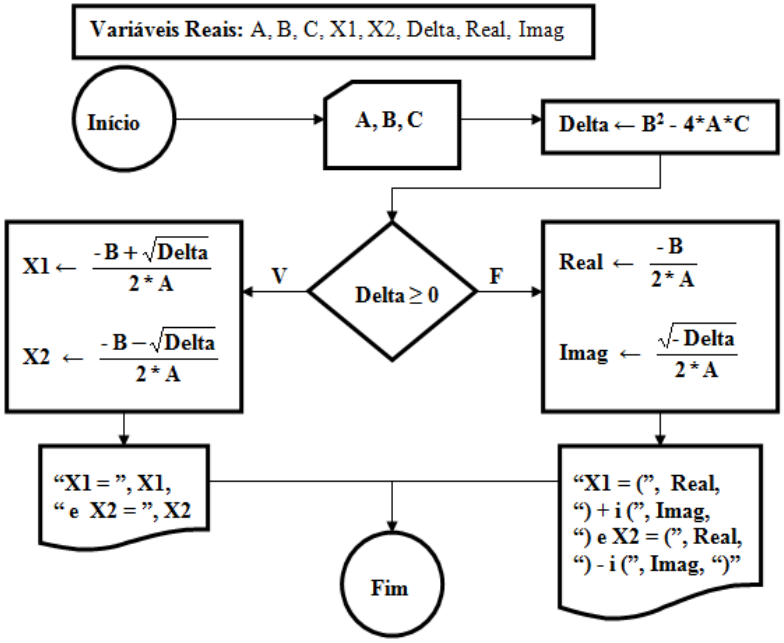
write ("X1 = (",  Real,  ")+i(",  Imag,  ") e X2 = (",  Real,  ")-i(",
        Imag,  ")");
    printf ("\nX1 = (%g)+i(%g) e X2 = (%g)-i(%g)", Real, Imag,
        Real, Imag);
```



Programa final

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    printf ("Digite os coeficientes de uma equacao de 2o grau: ");
    scanf ("%f%f%f", &A, &B, &C);
    Delta = pow(B,2) - 4*A*C;
    if (Delta >= 0) {
        X1 = (-B + sqrt(Delta)) / (2*A);
        X2 = (-B - sqrt(Delta)) / (2*A);
        printf ("\nX1 = %g e X2 = %g", X1, X2);
    }
    else {
        Real = -B / (2*A);
        Imag = sqrt(-Delta) / (2*A);
        printf ("\nX1 = (%g)+i(%g) e X2 = (%g)-i(%g)", Real, Imag, Real, Imag);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Livro 02



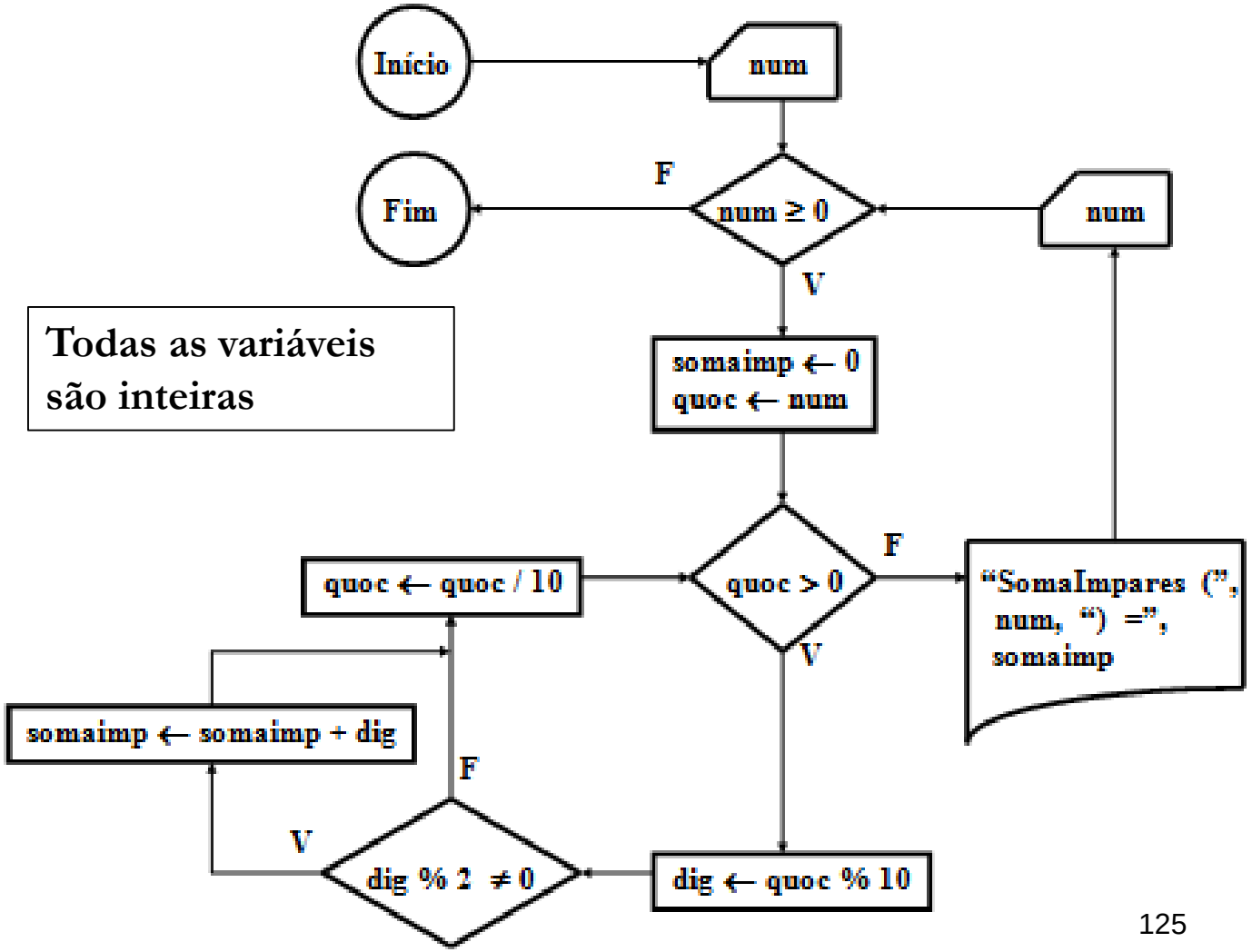
Salvar e executar

Exercício 4.1: Escrever e executar um **programa em C** para o seguinte **fluxograma** destinado a calcular a **soma dos dígitos ímpares** de vários números positivos lidos:

O programa deve ser amigável

É necessário discernir qual o comando correspondente a cada losango:

if-sem-else
if-else
while
do-while



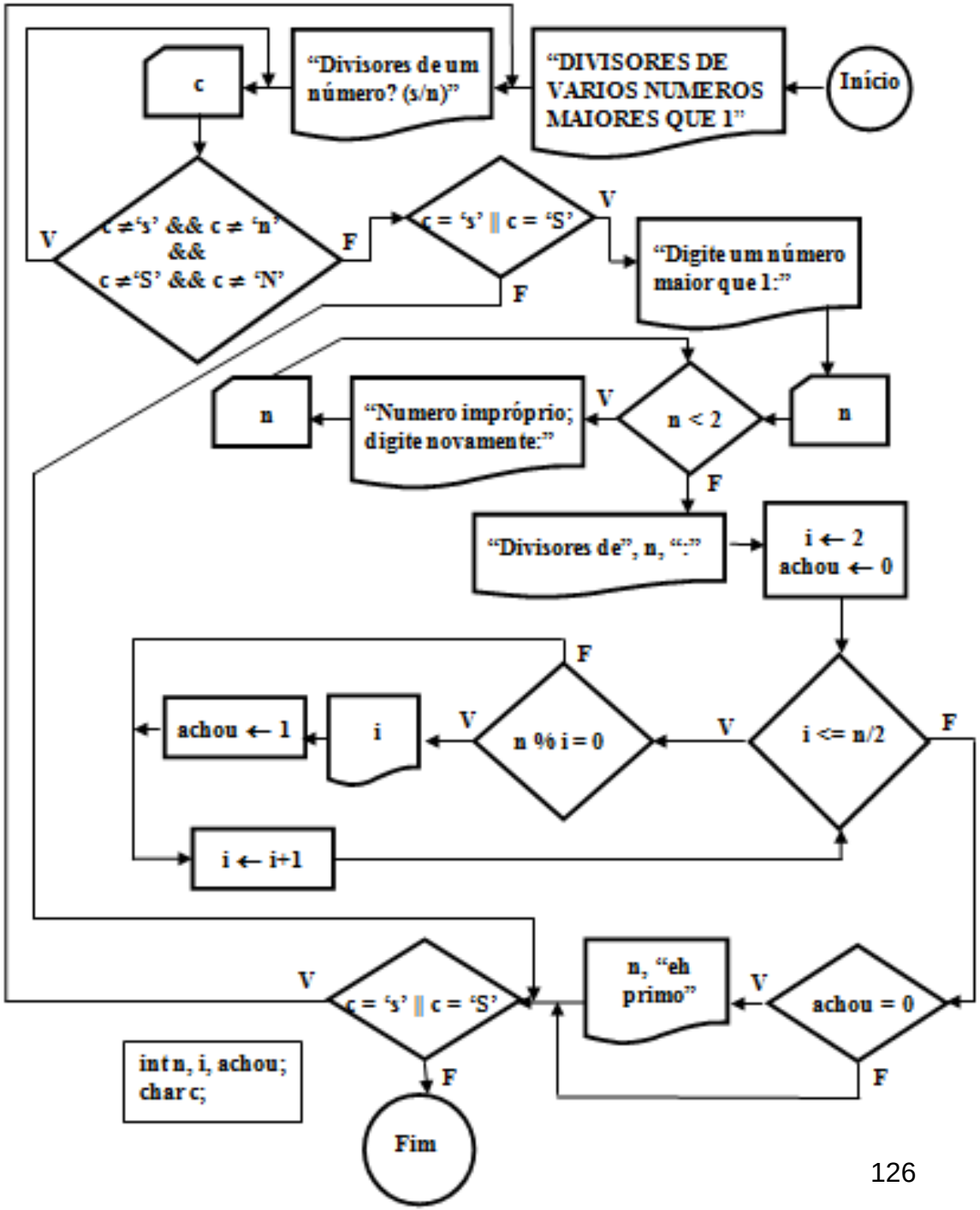
4º Exercício Laboratorial:

a) Fluxograma para encontrar os divisores de vários números lidos:

Escrever e executar um programa para ele

Cuidado para não confundir "=" com "=="

Escolher o comando correto para cada estrutura com losango



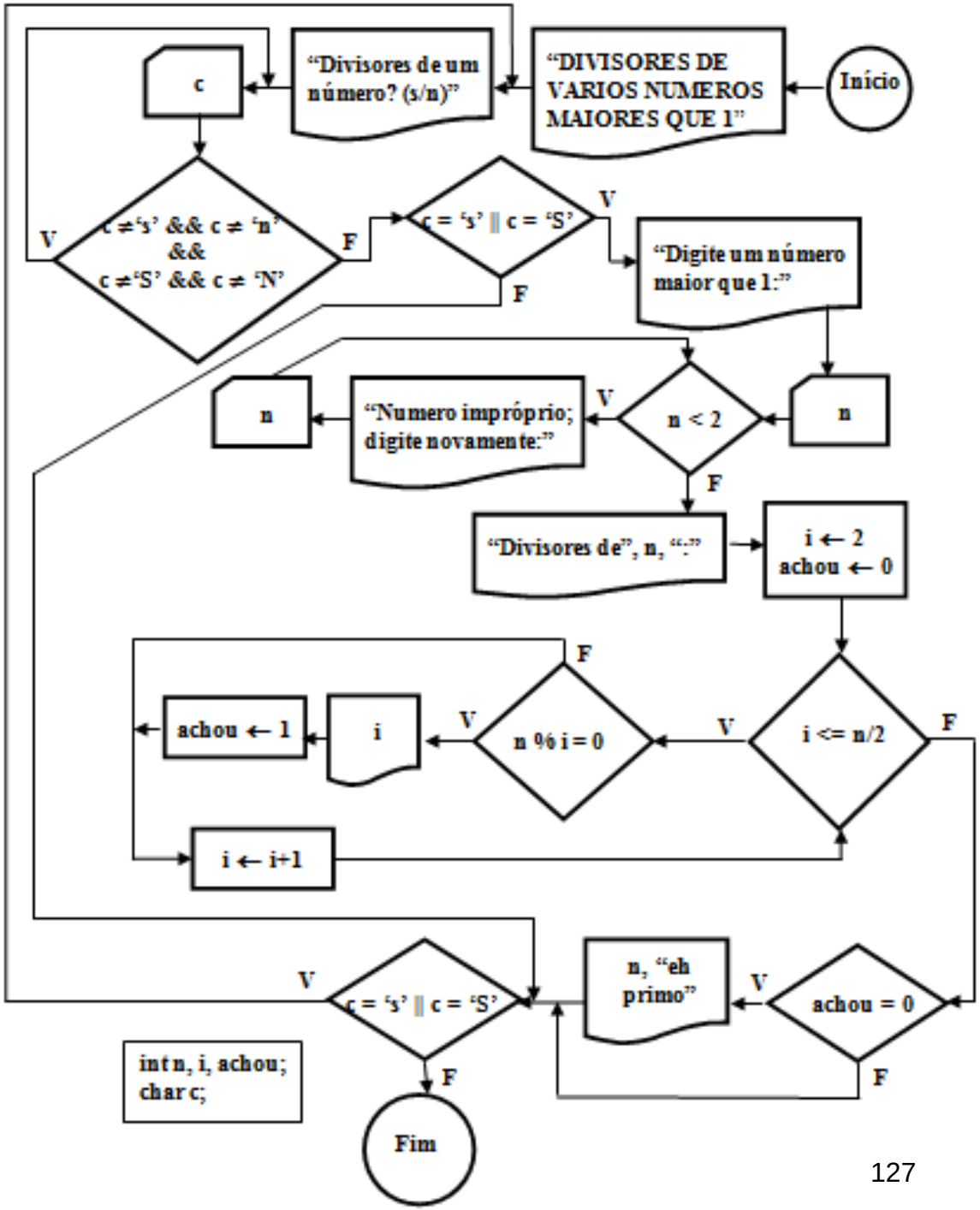
4º Exercício Laboratorial:

a) Fluxograma para encontrar os divisores de vários números lidos:

Esse comando não deve desobedecer ao fluxograma, mesmo que o programa funcione corretamente

No fluxograma não aparece o fechamento da tela

Deixar a saída no vídeo com certa estética



4º Exercício Laboratorial:

b) fluxograma para encontrar os MMC's de vários pares de números lidos

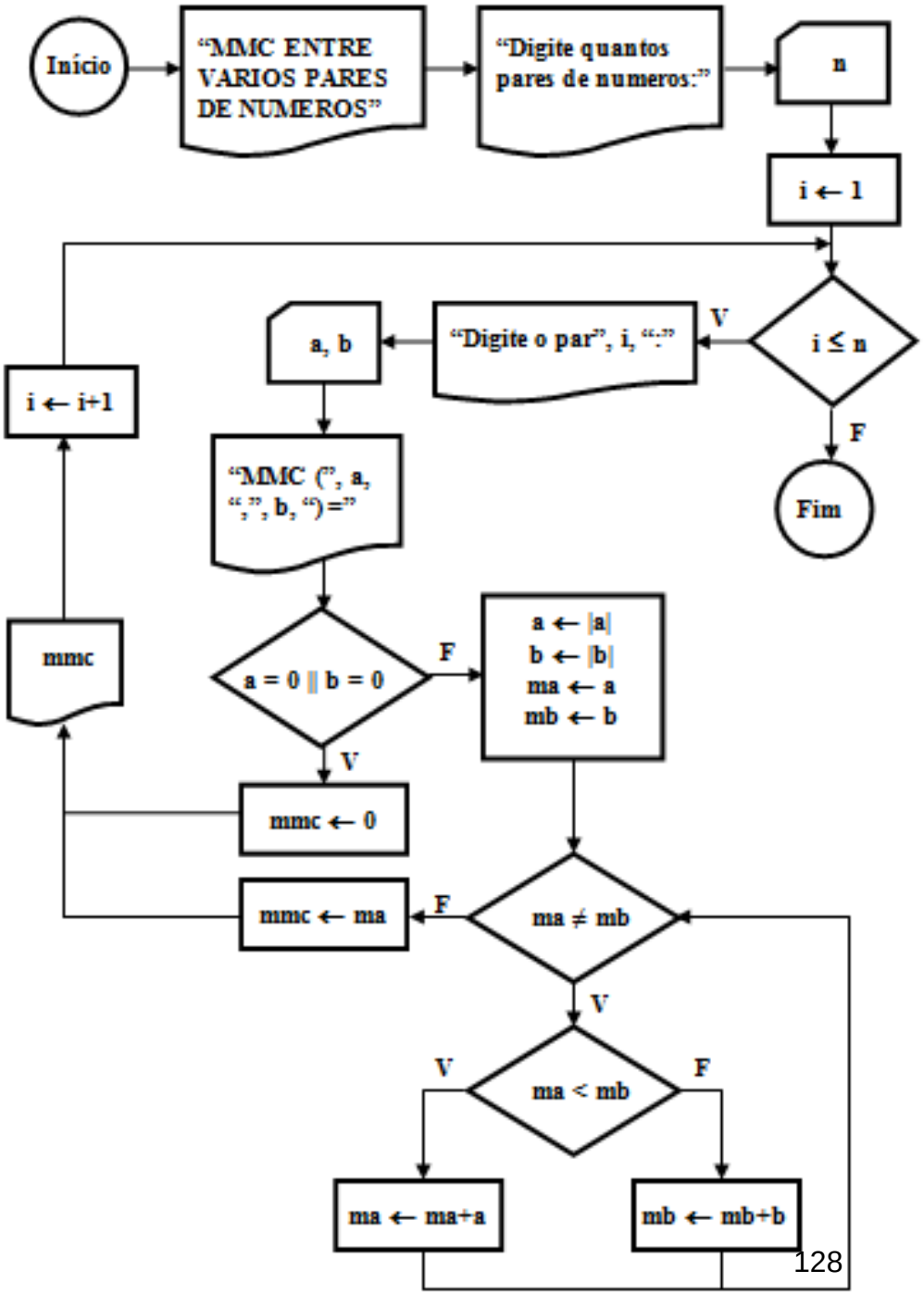
Escrever e executar um programa para este fluxograma

Valem os mesmos requisitos anteriores para escolher entre os comandos

- **if-sem-else, if-else,**
- **while e do-while**

No fluxograma não aparece o fechamento da tela

Deixar a saída no vídeo com certa estética



AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo V

Comandos For e Switch-Case

Outro comando repetitivo: for

Sejam os seguintes comandos:

```
int n, i, num, soma;
scanf ("%d", &n);
soma = 0; i = 1;
while (i <= n) {
    scanf ("%d", &num);
    soma += num;
    i++;
}
```

```
soma += num;

        substitui

soma = soma + num;


i++;      substitui

i = i + 1;
```

Outro comando repetitivo: for

Sejam os seguintes comandos:

```
int n, i, num, soma;
scanf ("%d", &n);
soma = 0; i = 1;
while (i <= n) {
    scanf ("%d", &num);
    soma += num;
    i++;
}
```

Eles podem ser escritos usando-se o comando for:

```
scanf ("%d", &n);  soma = 0;
for (i = 1; i <= n; i++) {
    scanf ("%d", &num);
    soma += num;
}
```

ou:

```
scanf ("%d", &n);
for (soma = 0, i = 1; i <= n; i++) {
    scanf ("%d", &num);
    soma += num;
}
```

Ou outras formas

- **Forma geral do comando for:**

  for (Inicializações ; Expressão ; Atualizações)
  Comando

- **Inicializações** e **Atualizações:**

  - Listas de zero ou mais **CmdAtribuição's**, **CmdEntrada's**, **CmdSaída's** e/ou **ChamadaFunção's**

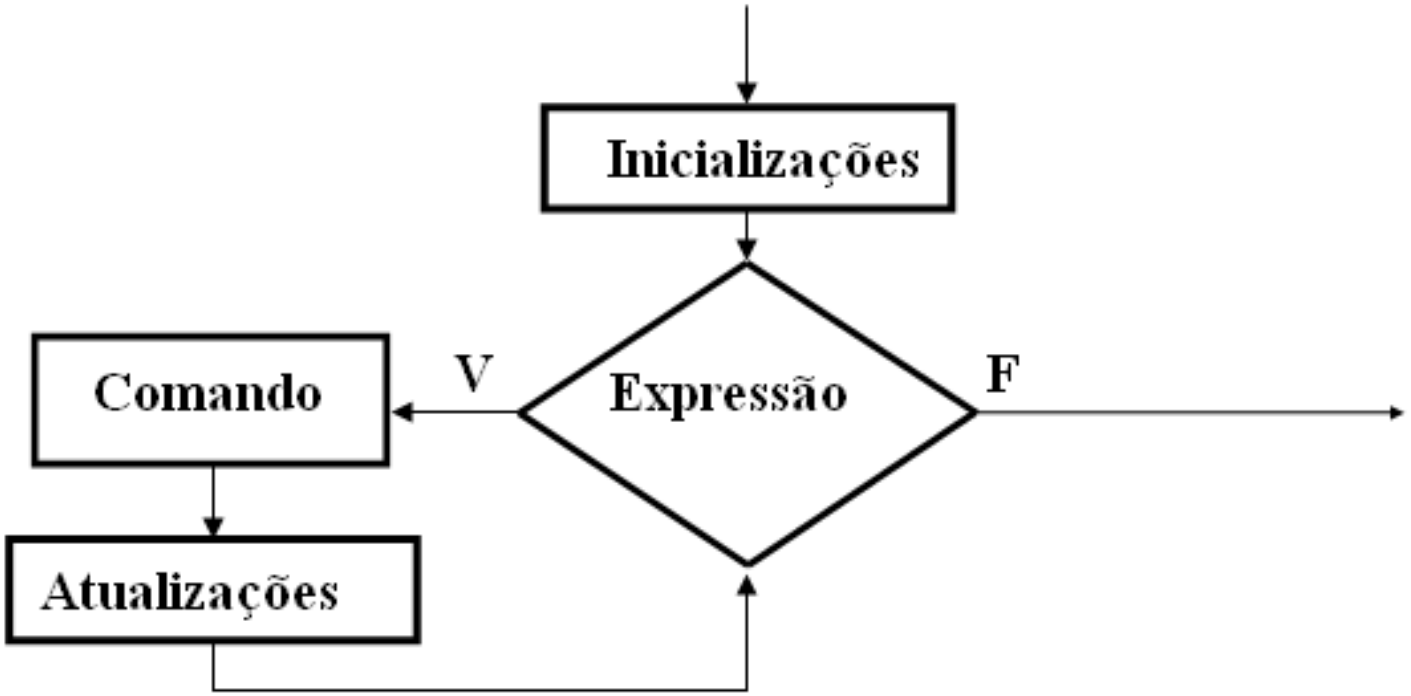
  - Separados por **vírgula** e não por **ponto e vírgula**

- O **CmdFor** acima equivale a

```
Inicializações
while ( Expressão ) {
        Comando
        Atualizações
}
```

Fluxograma do comando for:

- **for (Inicializações ; Expressão ; Atualizações) Comando**



Livro 02

Programa 5.1: Números perfeitos

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int n, i, div, soma;
    printf ("Digite um numero inteiro positivo: "); scanf ("%d", &n);
    printf ("\nNumeros perfeitos entre 1 e %d:\n\n", n);
    i = 1;
    while (i <= n) {
        soma = 0; div = 1;
        while (div*2 <= i) {
            if (i % div == 0) soma += div;
            div++;
        }
        if (soma == i) printf ("%12d", i);
        i++;
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

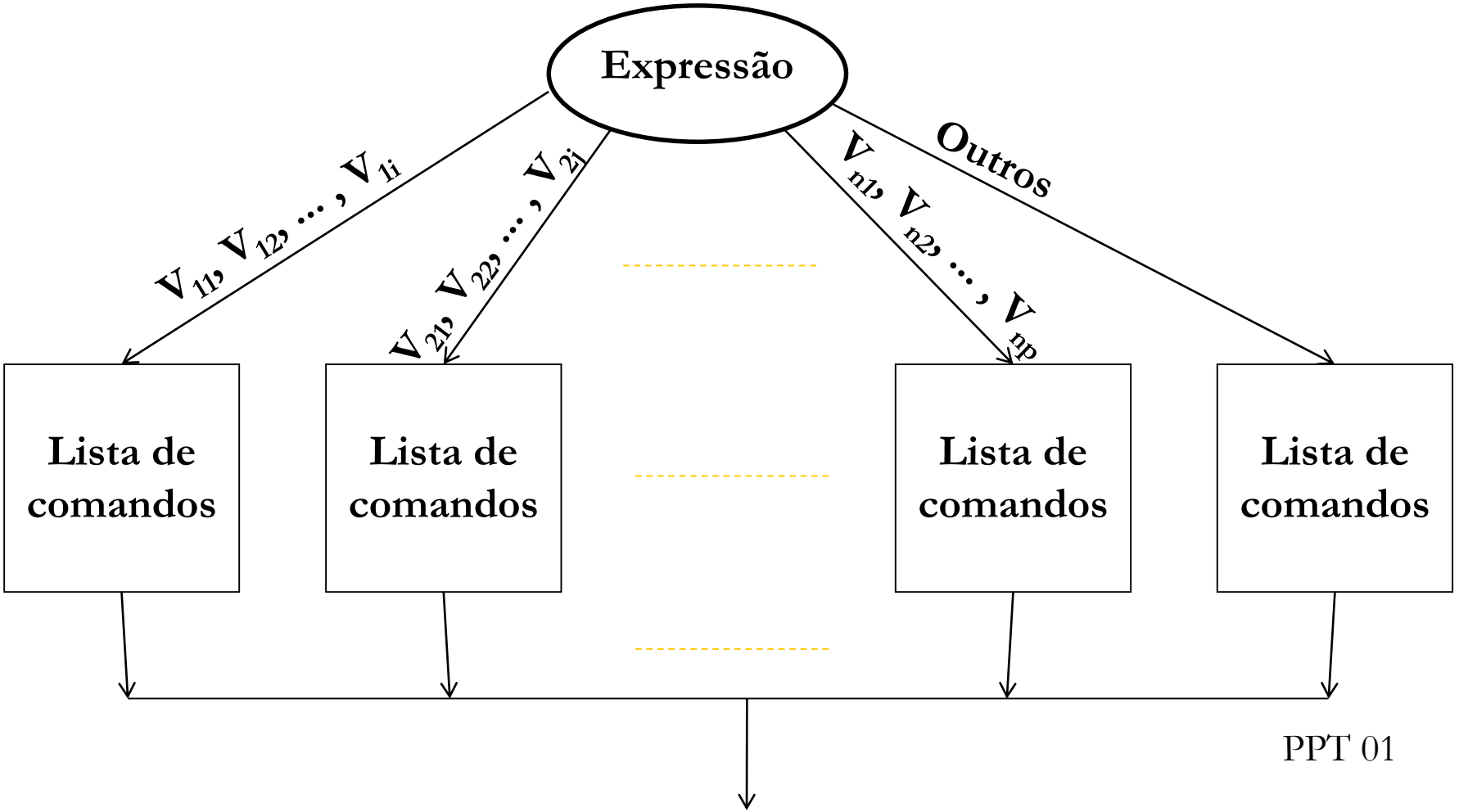
Número perfeito: a soma de seus divisores próprios é igual a si mesmo. Exemplos:

6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14

Usar comandos for em vez de comandos while

Salvar e executar

Comando condicional de várias alternativas:



PPT 01

```
#include <stdio.h>
#include <stdlib.h>
int main () {
        char c, lixo; int n;
        printf ("Levantar o astral? (s/n): "); scanf ("%c", &c);
        while (c == 's' || c == 'S') {
            printf ("\n\tDigite um numero de 1 a 7: "); scanf ("%d%c", &n, &lixo);
            switch (n) {
                        case 1: printf ("\n\tVoce eh inteligente"); break;
                        case 2: printf ("\n\tVoce eh campeao"); break;
                        case 3: printf ("\n\tVoce estah com tudo"); break;
                        case 4: printf ("\n\tSeu destino eh vencer"); break;
                        case 5: printf ("\n\tTudo coopera para seu sucesso"); break;
                        case 6: printf ("\n\tA vida sorri para voce"); break;
                        case 7: printf ("\n\tNada pode impedir sua vitoria"); break;
                        default: printf ("\n\tO numero deve ser de 1 a 7");
            }
            printf ("\n\nLevantar o astral? (s/n): "); scanf ("%c", &c);
        }
        printf ("\n\n"); system ("pause"); return 0;
}
```

Programa 5.2: Levantar o astral

Salvar e executar

Trocar o esquema while por for e executar

Eliminar alguns ou todos os break's e executar

Programa 5.3: Menu de operações aritméticas

```
/************************************
 *  Programa que oferece um menu de  *
 *  operacoes aritmeticas            *
 ************************************/

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

/*  Cabecalho, declaracoes e oferecimento de servico  */

int main () {       // Inicio da funcao main
   char c, opcao, lixo; float n1, n2, result;
   printf ("C A L C U L O S    A R I T M E T I C O S");
   printf ("\n\nResolver algum calculo (s/n)? ");
   scanf ("%c%c", &c, &lixo);
   while (c == 's' || c == 'S') {
```

```
/*  Apresentacao de um menu de calculos aritmeticos  */

    printf ("\n\nDigitar:");
    printf ("\n\n\tS ou s para soma");
    printf ("\n\tT ou t para subtracao");
    printf ("\n\tM ou m para multiplicacao");
    printf ("\n\tD ou d para divisao");
    printf ("\n\tQ ou q para elevar ao quadrado");
    printf ("\n\tR ou r para extrair a raiz quadrada");
    printf ("\n\nSua opcao: "); scanf ("%c%c", &opcao, &lixo);
```

```
switch (opcao) {
        case 'S': case 's':  // Soma de 2 numeros
                printf ("\n\tDigite os operandos: ");
                scanf ("%f%f%c", &n1, &n2, &lixo);
                result = n1 + n2;
                printf ("\n\tResultado: %g", result);
                break;
        case 'T': case 't': // Subtracao entre 2 numeros
                printf ("\n\tDigite os operandos: ");
                scanf ("%f%f%c", &n1, &n2, &lixo);
                result = n1 - n2;
                printf ("\n\tResultado: %g", result);
                break;
        case 'M': case 'm': // Multiplicacao de 2 numeros
                printf ("\n\tDigite os operandos: ");
                scanf ("%f%f%c", &n1, &n2, &lixo);
                result = n1 * n2;
                printf ("\n\tResultado: %g", result);
                break;
```

Obs.: mais de um case para o mesmo código

```
        case 'D': case 'd':  // Divisao entre 2 numeros
                printf ("\n\tDigite os operandos: ");
                scanf ("%f%f%c", &n1, &n2, &lixo);
                if (n2 == 0)
printf ("\n\tErro: tentativa de divisao por zero");
                else {
                        result = n1 / n2;
                        printf ("\n\tResultado: %g", result);
                }
                break;
        case 'Q': case 'q':  // Quadrado de um numero
                printf ("\n\tDigite o operando: ");
                scanf ("%f%c", &n1, &lixo);
                result = n1 * n1;
                printf ("\n\tResultado: %g", result);
                break;
```

```
        case 'R': case 'r':  // Raiz quadrada de um numero
                printf ("\n\tDigite o operando: ");
                scanf ("%f%c", &n1, &lixo);
                if (n1 < 0)
        printf ("\n\tErro: sqrt de numero negativo");
                else {
                        result = sqrt (n1);
                        printf ("\n\tResultado: %g", result);
                }
                break;
        default:          // Erro na digitacao da opcao
                printf ("\n\tErro na opcao");
}
```

```
/*  Novo oferecimento de servico  */

      printf ("\n\nResolver outro calculo (s/n)? ");
      scanf ("%c%c", &c, &lixo);
   }

/*  Fechamento da tela  */

   printf ("\n\n"); system ("pause"); return 0;

}       //Fim da funcao main
```

Copiar, salvar e executar

Programa 5.4: Implementação de um autômato finito

- **Autômato finito:** modelo matemático reconhecedor de cadeias de caracteres com determinadas características relativamente simples

- Também conhecido como **máquina de estados finitos**

- **Exemplos de características:**

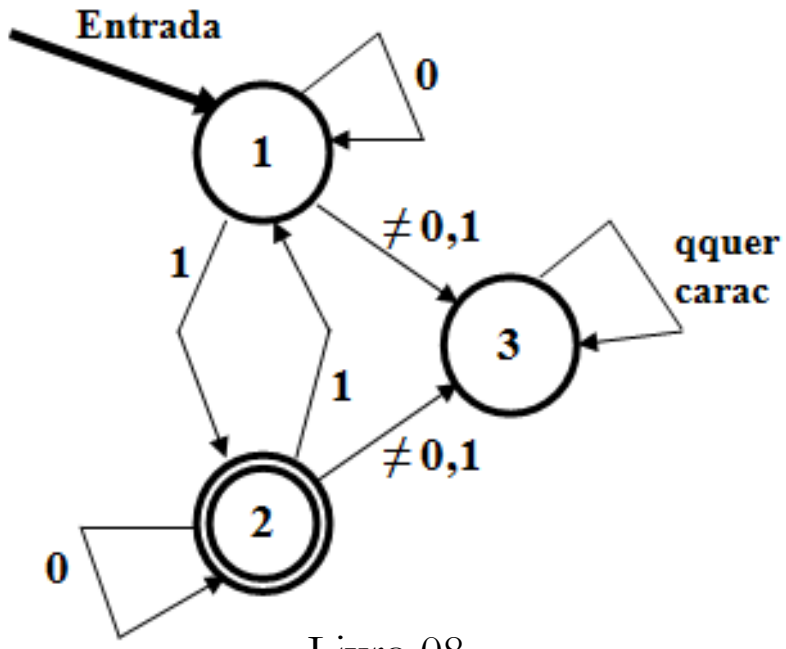
  - Cadeias binárias (contendo apenas os dígitos **0** e **1**)
  - Cadeias binárias em que o número de dígitos **1** é impar
  - Cadeias contendo comentários da Linguagem C

- Para características mais complexas existem **autômatos com pilha**, **máquinas de Turing**, e outros tipos de autômatos

Programa 5.4: Implementação de um autômato finito

- Autômatos finitos podem ser ilustrados por **diagramas de transições de estados**

- **Exemplo:** autômato reconhecedor de cadeias binárias em que o número de dígitos **1** é impar



Livro 08

Percorre-se o autômato usando os caracteres da cadeia

Estado 1: o número de 1's é par

Estado 2: o número de 1's é impar

Estado 3: cadeia não binária

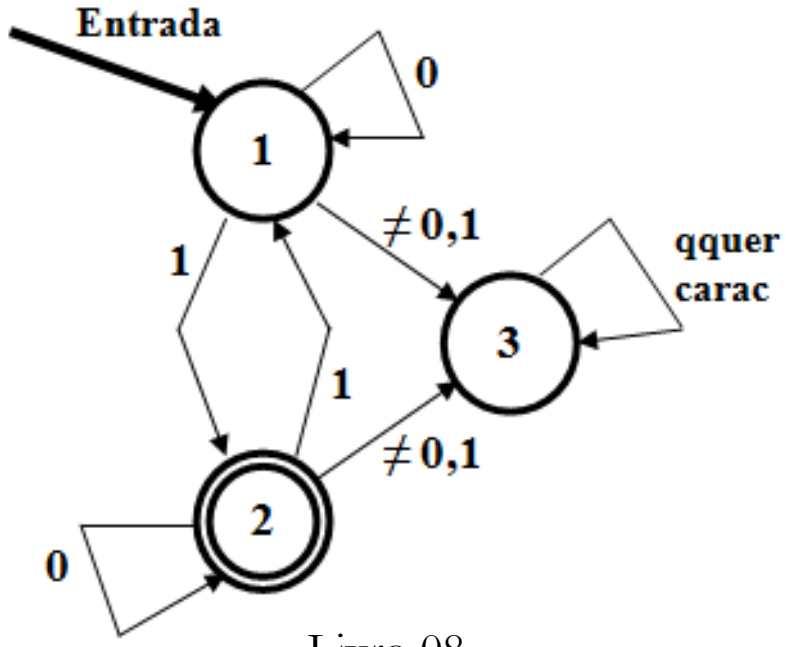
Círculo duplo: estado de aceitação (Estado 2)

A cadeia só é aceita se seu percurso terminar no estado de aceitação

Programa 5.4: Implementação de um autômato finito

- Autômatos finitos podem ser ilustrados por **diagramas de transições de estados**

- **Exemplo:** autômato reconhecedor de cadeias binárias em que o número de dígitos **1** é impar

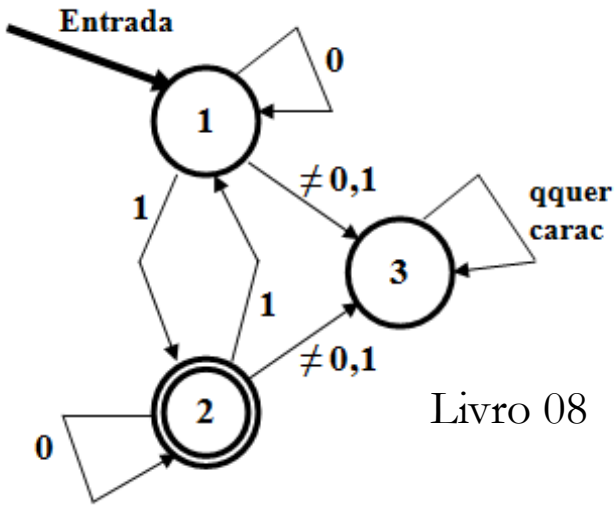


Um autômato pode ter mais de um estado de aceitação

O estado de entrada é chamado de estado inicial

Um estado de aceitação também é chamado de estado final

Seja a seguir a implementação deste autômato em Linguagem C



Livro 08

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

/* Cabecalho, declaracoes e digitação do numero de cadeias */

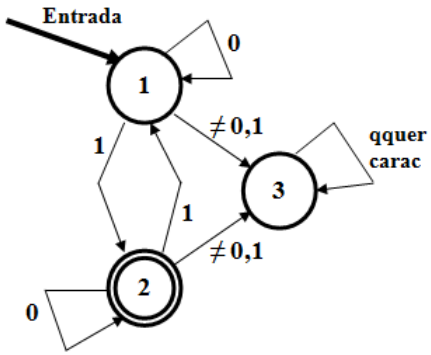
int main () {   //  Inicio da funcao main
   char c, carac, lixo; int estado, i, n;
   printf ("N U M E R O   I M P A R   D E   U M'S");
   printf ("\n\nNumero de cadeias a serem testadas: ");
   scanf ("%d%c", &n, &lixo);
   for (i = 1; i <= n; i++) {

/* Digitacao da cadeia a ser testada */

      printf ("\n\nDigite a %da cadeia encerrada por <enter>: ", i);
```

```
/* Percurso no automato usando os caracteres da cadeia */

        for (estado = 1, scanf ("%c", &carac); carac != '\n';
                                                scanf ("%c", &carac))
            switch (estado) {
                case 1:          // Teste do caractere no estado 1
                    switch (carac) {
                        case '0': estado = 1; break;
                        case '1': estado = 2; break;
                        default: estado = 3;
                    }
                    break;
                case 2:          // Teste do caractere no estado 2
                    switch (carac) {
                        case '0': estado = 2; break;
                        case '1': estado = 1 ; break;
                        default: estado = 3;
                    }
                    break;
                case 3:          // Acao para o estado 3 (dispensavel)
                    estado = 3;
                    break;
            }
```



Livro 08

```
/* Escrita do resultado do teste */

        if (estado == 2)
                printf ("\n\tCadeia aprovada!");
        else
                printf ("\n\tCadeia reprovada!");
   }

/* Fechamento da tela */

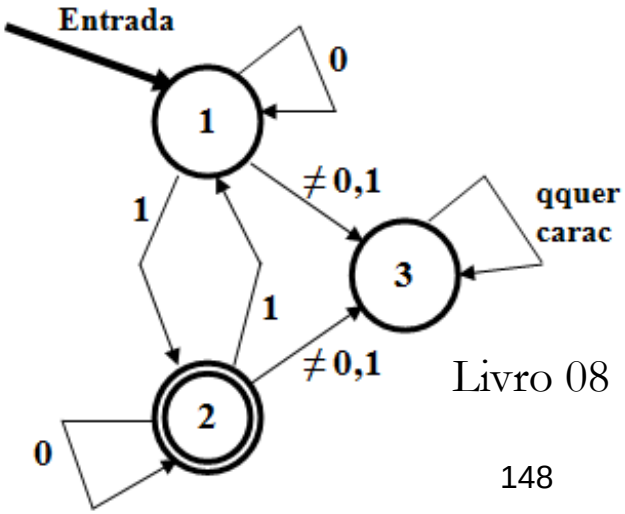
   printf ("\n\n"); system ("pause");
   return 0;

}        //Fim da funcao main e do programa
```

Copiar, salvar e executar

Alterar o autômato e o programa para aceitar apenas cadeias com número par de 1's

E se a contagem fosse de dígitos 0's?

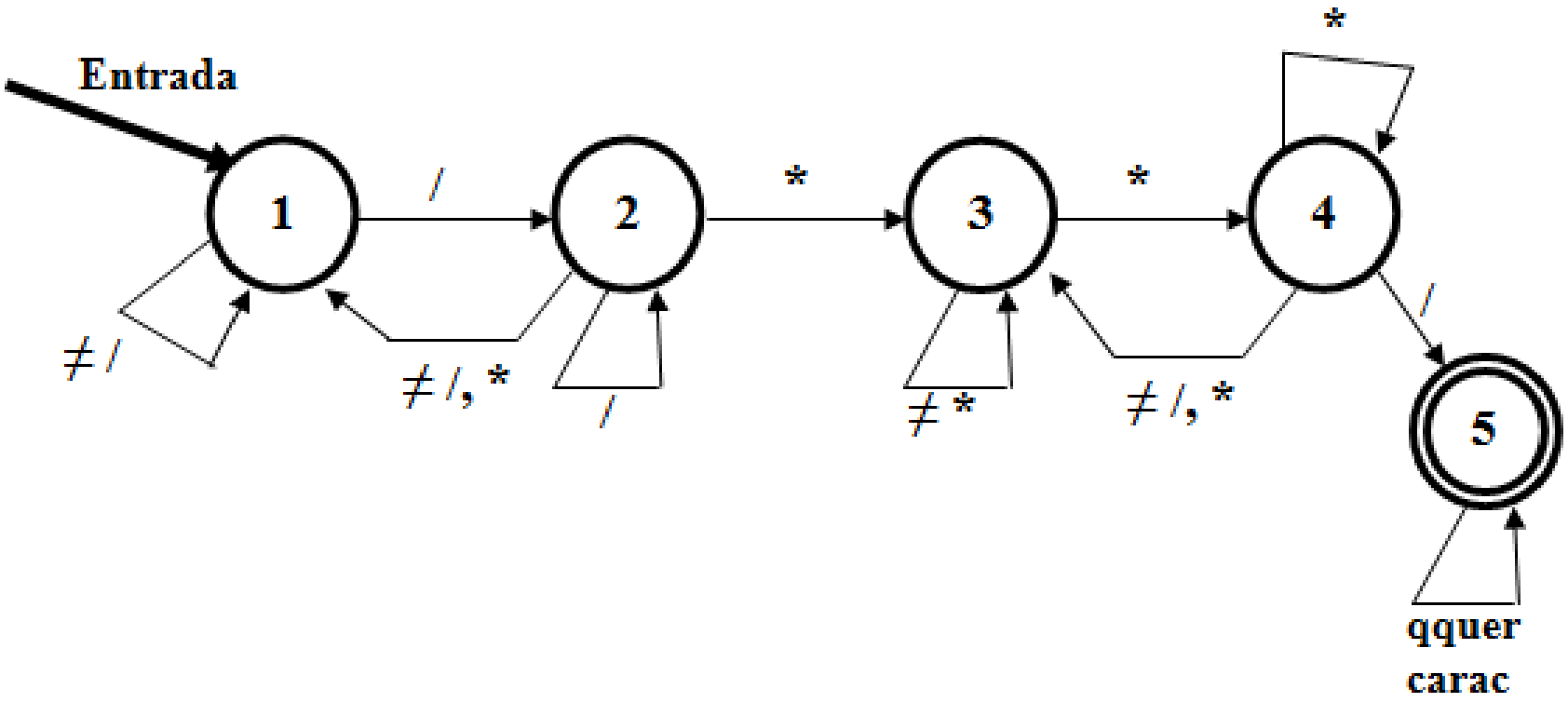


Aplicações de autômatos finitos:

- Estudo de linguagens e compiladores para computadores
- Elaboração de circuitos eletrônicos lógicos
- Mecanismos de busca na web
- Automação de projetos eletrônicos
- Dispositivos eletromecânicos do dia a dia
- Calculadoras
- Inteligência artificial
- Sistemas neurológicos

5º Exercício Laboratorial:

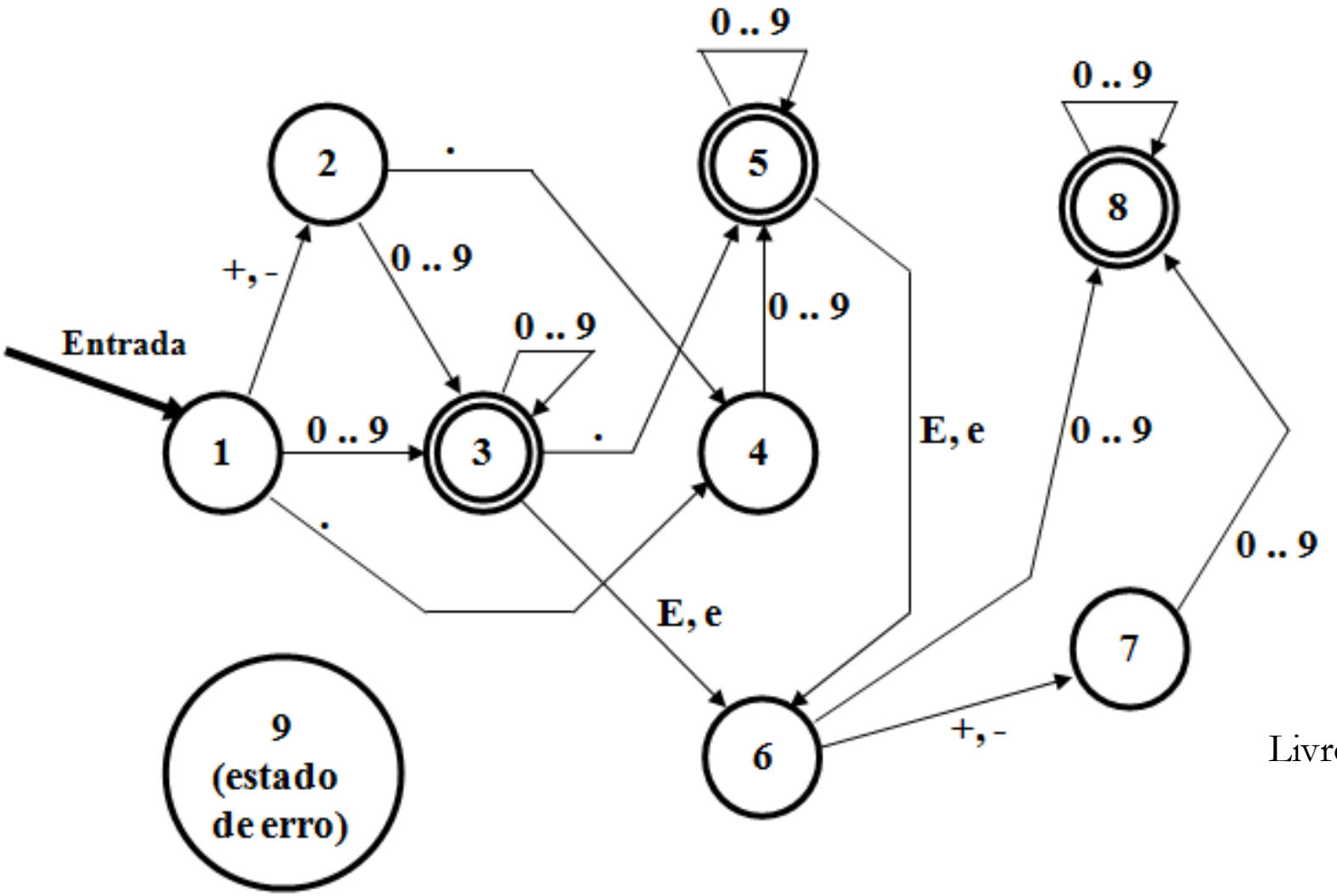
a) Autômato reconhecedor de cadeias contendo comentários (/* ..... */)



Livro 08

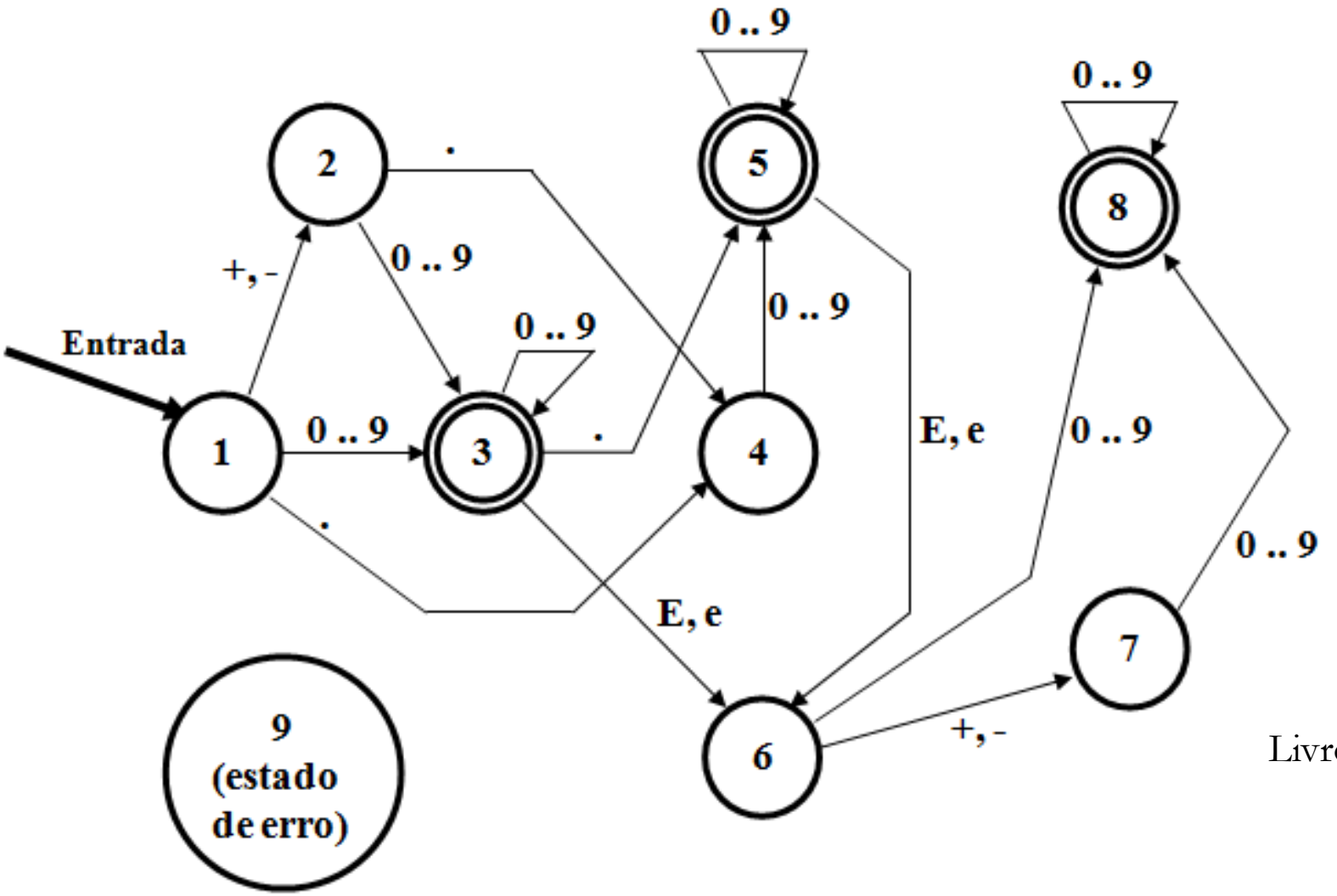
5º Exercício Laboratorial:

b) autômato reconhecedor de constantes numéricas em C



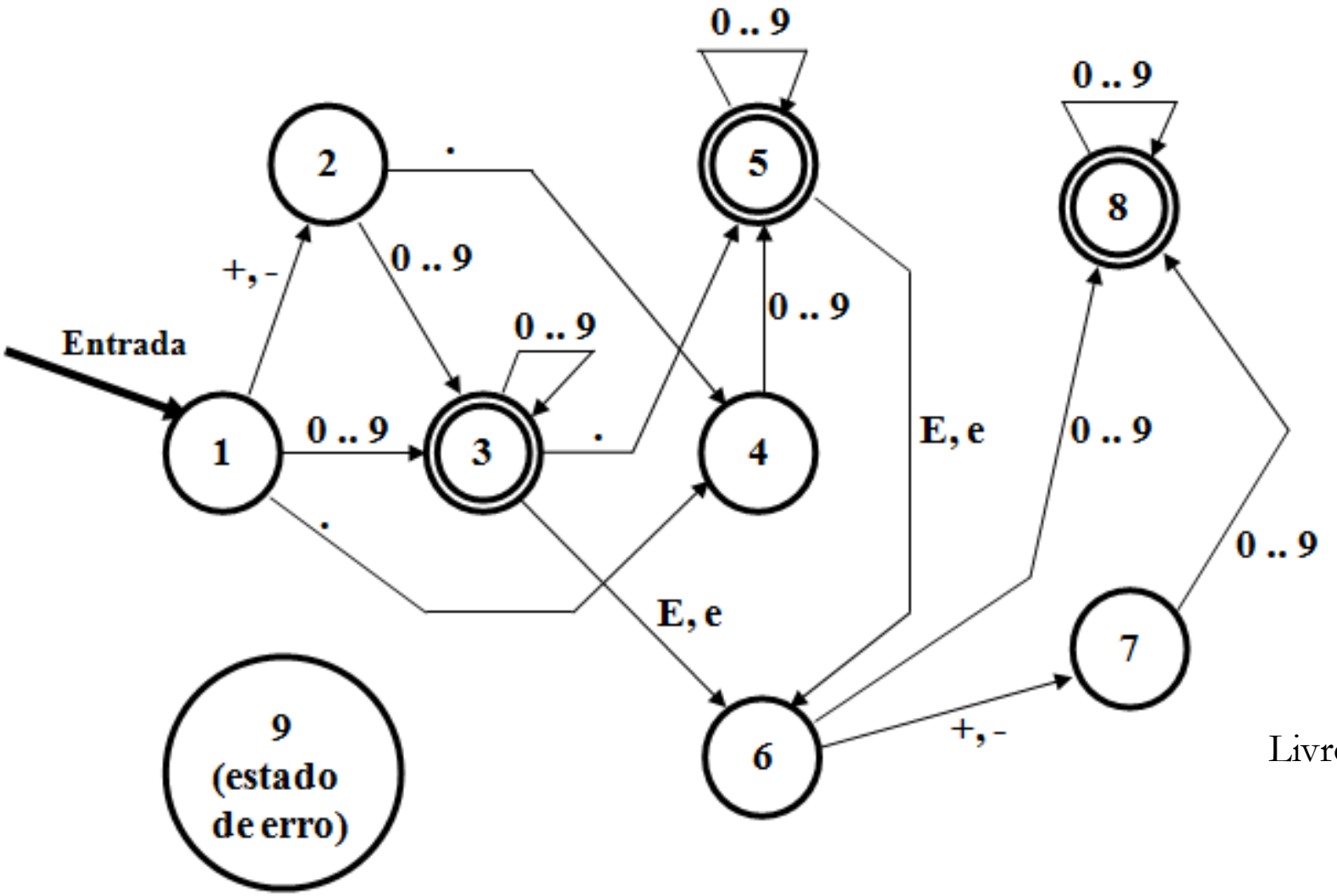
Livro 08

- Existe uma seta de todos os estados para o estado **9**, inclusive desse último para ele mesmo



Livro 08

- Devem ser reconhecidas constantes como: **13**, **+13**, **-13**, **13.**, **1.25**, **.25**, **-.25**, **-32.43**, **13E-15**, **13.e-15**, **-13.25E+72**, **.75e5**



Livro 08

5º Exercício Laboratorial:

c) autômato reconhecedor de cadeias binárias contendo somente dígitos 0's e 1's alternados

- As cadeias não podem ter **dígitos iguais seguidos**
- A **cadeia vazia** não deve ser aceita, mas cadeias com **apenas um dígito** devem ser aceitas
- Primeiramente deve ser projetado o **diagrama** do autômato
- Em seguida, o **programa** deve ser escrito, fundamentado nesse **diagrama**

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo VI

Desenhos e Animação

Arquivos fornecidos para a aula:

- **conio2.zip:** arquivos para a biblioteca **conio2** a ser utilizada no **CodeBlocks** (Aba **Códigos** da página do Prof Fábio Mokarzel); contém:
  - **conio2.h** e **libconio.a**

- Página do Prof. Fábio Mokarzel:
  - **http://www.comp.ita.br/~mokarzel**

Instruções para usar o conio2 no CodeBlocks:

- Adaptação de documento do Prof. Paulo André Lima de Castro

1. Copiar o arquivo **conio2.h** para **C:\Arquivos de Programas (x86)\CodeBlocks\MingW\include**

2. Copiar o arquivo **libconio.a** para **C:\Arquivos de Programas (x86)\CodeBlocks\MingW\lib**

3. Abrir o **CodeBlocks** e entrar no menu **Settings → Compiler**

Instruções para usar o conio2 no CodeBlocks:

4. Clicar no botão **"Linker Settings"**

5. Abaixo do quadro **"Link libraries"**, clicar no botão **"Add"**

6. Escrever na janelinha **"Add library"** o seguinte: **libconio.a**

7. No programa, colocar: **#include <conio2.h>**

Algumas funções do conio2:

textbackground () – Seta a cor de fundo
textcolor () – Seta a cor do texto
clrscr () – Limpa a tela
gotoxy () – Coloca o cursor em uma posição da tela

Programa 6.1: Posicionamento do cursor no vídeo-texto

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int col, lin; char c;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   printf ("Posicionar cursor? (s/n): ");
   c = getche();
   while (c == 's' || c == 'S') {
       printf ("\rDigite a coluna e a linha: ");
       scanf ("%d%d", &col, &lin);
       gotoxy (col, lin); getche ();
       gotoxy (1, 1);
       printf ("                                                  ");
       printf ("\rPosicionar cursor? (s/n): "); c = getche();}
   return 0;
}
```

clrscr (conio2.h): limpa a tela

\r: reescreve a 1ª linha da tela

Posiciona o cursor nas coordenadas desejadas – digitar um caractere

Programa 6.1: Posicionamento do cursor no vídeo-texto

Copiar, salvar e executar

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int col, lin; char c;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   printf ("Posicionar cursor? (s/n): ");
   c = getche();
   while (c == 's' || c == 'S') {
       printf ("\rDigite a coluna e a linha: ");
       scanf ("%d%d", &col, &lin);
       gotoxy (col, lin); getche ();
       gotoxy (1, 1);
       printf ("                                        ");
       printf ("\rPosicionar cursor? (s/n): "); c = getche();}
   return 0;
}
```

Posiciona o cursor na coluna e linha 1

Limpa a 1ª linha

\r: reescreve a 1ª linha da tela

Programa 6.2: Determinação dos limites do gotoxy

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int i, n;
   printf ("TESTE DO LIMITE DE LINHAS DO GOTOXY");
   printf ("\n\n\tDigite o numero de linhas: ");
   scanf ("%d", &n);
   printf ("\nMaximize a tela e digite algo!");
   getche (); clrscr ();
   for (i=1; i<=n; i++) {
       gotoxy ((i-1)%80+1, i); printf ("%d", i%10);
   }
   printf ("\nDigite algo para encerrar: ");
   getch ();
   return 0;
}
```

Copiar, salvar e executar

Colunas variam de 1 a 80

Valores escritos variam de 0 a 9

Experimentar os seguintes valores para n:
40, 53, 60, 80, 200, 300, 400

Qual o limite de linhas atingidas pelo gotoxy?
Qual o número de linhas da tela maximizada?

Programa 6.3: Passeio do '*' pelo vídeo-texto

Copiar, salvar
e executar

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int i, j, demora = 20000000;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   for (i = 10; i <= 60; i++) {
       gotoxy (i, 5); printf (" *");
       for (j = 1; j <= demora; j++);}
   for (i = 6; i <= 40; i++) {
       gotoxy (61, i-1); printf (" ");
       gotoxy (61, i); printf ("*");
       for (j = 1; j <= demora; j++);
   }
   printf ("\n\nDigite algo para encerrar: ");
   getch (); return 0;
}
```

Programa 6.3: Passeio do '*' pelo vídeo-texto

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int i, j, demora = 20000000;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   for (i = 10; i <= 60; i++) {
        gotoxy (i, 5); printf (" *");
        for (j = 1; j <= demora; j++);}
   for (i = 6; i <= 40; i++) {
        gotoxy (61, i-1); printf (" ");
        gotoxy (61, i); printf ("*");
        for (j = 1; j <= demora; j++);
   }
   printf ("\n\nDigite algo para encerrar: ");
   getch (); return 0;
}
```

Para i = 10, na linha 5:

Escreve ' ' na coluna 10 e '*' na coluna 11

Para i = 11, na linha 5:

Escreve ' ' na coluna 11, apagando '*'

Escreve '*' na coluna 12

Dá a impressão de movimento do '*' uma posição p/direita

Para i = 60, '*' vai para a coluna 61

Programa 6.3: Passeio do '*' pelo vídeo-texto

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int i, j, demora = 20000000;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   for (i = 10; i <= 60; i++) {
        gotoxy (i, 5); printf (" *");
        for (j = 1; j <= demora; j++); }
   for (i = 6; i <= 40; i++) {
        gotoxy (61, i-1); printf (" ");
        gotoxy (61, i); printf ("*");
        for (j = 1; j <= demora; j++);
   }
   printf ("\n\nDigite algo para encerrar: ");
   getch (); return 0;
}
```

Faz '*' permanecer numa posição durante um Δt perceptível

O valor 20.000.000 para 'demora' é experimental

Esse valor pode variar com o computador

Programa 6.3: Passeio do '*' pelo vídeo-texto

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int i, j, demora = 20000000;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   for (i = 10; i <= 60; i++) {
        gotoxy (i, 5); printf (" *");
        for (j = 1; j <= demora; j++);}
   for (i = 6; i <= 40; i++) {
        gotoxy (61, i-1); printf (" ");
        gotoxy (61, i); printf ("*");
        for (j = 1; j <= demora; j++);
   }
   printf ("\n\nDigite algo para encerrar: ");
   getch (); return 0;
}
```

Para i = 6, na coluna 61:

Escreve ' ' na linha 5, apagando '*'

Escreve '*' na linha 6

Dá a impressão de movimento do '*' uma posição p/baixo

Para i = 40, '*' vai para a linha 40

Programa 6.3: Passeio do '*' pelo vídeo-texto

```
#include <stdio.h>
#include <conio.h>
#include <conio2.h>
int main () {
   int i, j, demora = 20000000;
   printf ("Maximize a tela e digite algo: ");
   getche (); clrscr ();
   for (i = 10; i <= 60; i++) {
        gotoxy (i, 5); printf (" *");
        for (j = 1; j <= demora; j++);}
   for (i = 6; i <= 40; i++) {
        gotoxy (61, i-1); printf (" ");
        gotoxy (61, i); printf ("*");
        for (j = 1; j <= demora; j++); <-------
   }
   printf ("\n\nDigite algo para encerrar: ");
   getch (); return 0;
}
```

Também faz '*' permanecer numa posição durante um Δt perceptível

6º Exercício Laboratorial: Escrever um programa em C para:

- Fazer um **desenho animado no vídeo-texto**; o tema do desenho é livre

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo VII

Variáveis Indexadas Numéricas

Programa 7.1: Vetor de 10 elementos nulos

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int i, V[10] = {0};
   printf ("Vetor de 10 elementos nulos:\n\n");
   system ("pause");
   printf ("\n\n");
   for (i = 0; i <= 9; i++)
       printf ("%4d", V[i]);
   printf ("\n\n"); system ("pause");
   return 0;
}
```



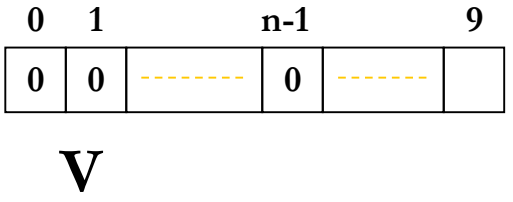
i varia de 0 a 9, escrevendo na mesma linha os valores de:

V[0], V[1], ... , V[9]

Copiar, salvar e executar

Programa 7.2: Vetor de n elementos nulos

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int i, n, V[10];
   printf ("Vetor de n elementos nulos:\n");
   do {
        printf ("\nDigite n (<= 10): ");
        scanf ("%d", &n);
   } while (n < 1 || n > 10);
   for (i = 0; i <= n-1; i++)
        V[i] = 0;
   printf ("\n");
   for (i = 0; i <= n-1; i++)
        printf ("%4d", V[i]);
   printf ("\n\n"); system ("pause"); return 0;
}
```

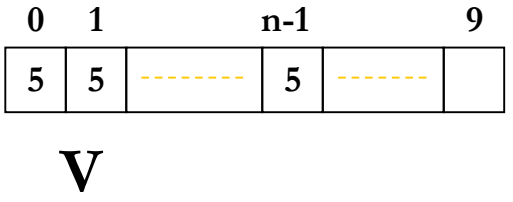


V

Copiar, salvar e executar

Programa 7.3: Vetor de n elementos 5

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int i, n, V[10];
   printf ("Vetor de n elementos iguais a 5:\n");
   do {
        printf ("\nDigite n (<= 10): ");
        scanf ("%d", &n);
   } while (n < 1 || n > 10);
   for (i = 0; i <= n-1; i++)
        V[i] = 5;
   printf ("\n");
   for (i = 0; i <= n-1; i++)
        printf ("%4d", V[i]);
   printf ("\n\n"); system ("pause"); return 0;
}
```

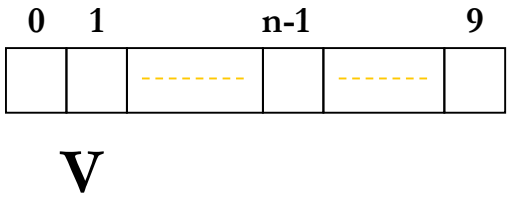


V

Copiar, salvar e executar

Programa 7.4: Leitura e escrita de vetor

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int i, n, V[10];
   printf ("Leitura e escrita de vetor\n\n");
   do {
        printf ("\nDigite o n.o de elementos (<= 10): ");
        scanf ("%d", &n);
   } while (n < 1 || n > 10);
   printf ("\n\n");
   for (i = 0; i <= n-1; i++) {
        printf ("Elemento %d: ", i);
        scanf ("%d", &V[i]);
   }
   printf ("\n\nVetor V:\n\n");
   for (i = 0; i <= n-1; i++)
        printf ("%6d", V[i]);
   printf ("\n\n"); system ("pause"); return 0;
}
```

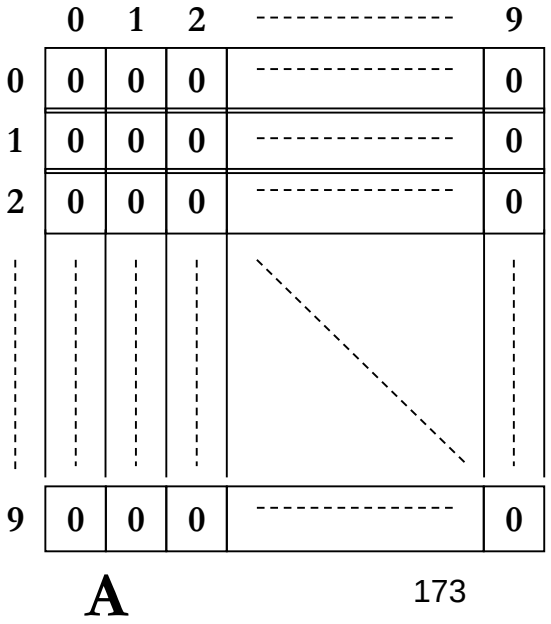


Copiar, salvar e executar

Programa 7.5: Matriz 10 x 10 nula

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10] = {0}, i, j;
   printf ("Matriz 10x10 nula:\n\n");
   system ("pause");
   printf ("\n\n");
   for (i = 0; i <= 9; i++) {
       for (j = 0; j <= 9; j++)
              printf ("%4d", A[i][j]);
       printf ("\n");
   }
   printf ("\n\n");
   system ("pause"); return 0;
}
```

Copiar, salvar e executar



A

Programa 7.5: Matriz 10 x 10 nula

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10] = {0}, i, j;
   printf ("Matriz 10x10 nula:\n\n");
   system ("pause");
   printf ("\n\n");
   for (i = 0; i <= 9; i++) {
        for (j = 0; j <= 9; j++)
              printf ("%4d", A[i][j]);
        printf ("\n");
   }
   printf ("\n\n");
   system ("pause"); return 0;
}
```

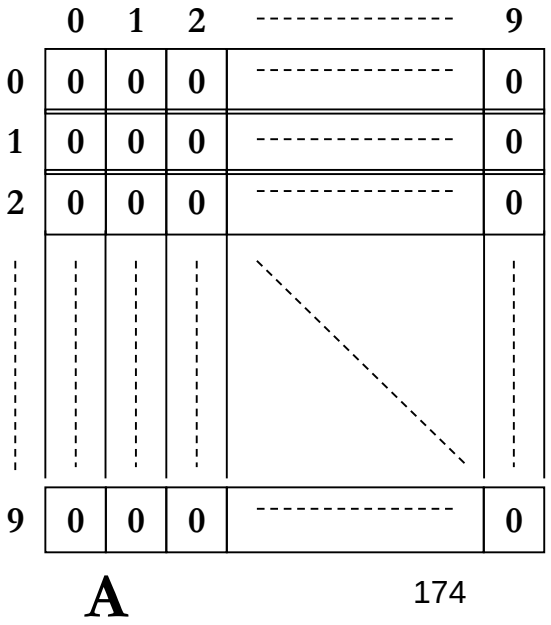
Para i = 0

j varia de 0 a 9, escrevendo na mesma linha os valores de:

A[0][0], A[0][1], ... , A[0][9]

Depois o cursor do vídeo vai para a linha seguinte

Copiar, salvar e executar



Programa 7.5: Matriz 10 x 10 nula

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10] = {0}, i, j;
   printf ("Matriz 10x10 nula:\n\n");
   system ("pause");
   printf ("\n\n");
   for (i = 0; i <= 9; i++) {
        for (j = 0; j <= 9; j++)
              printf ("%4d", A[i][j]);
        printf ("\n");
   }
   printf ("\n\n");
   system ("pause"); return 0;
}
```

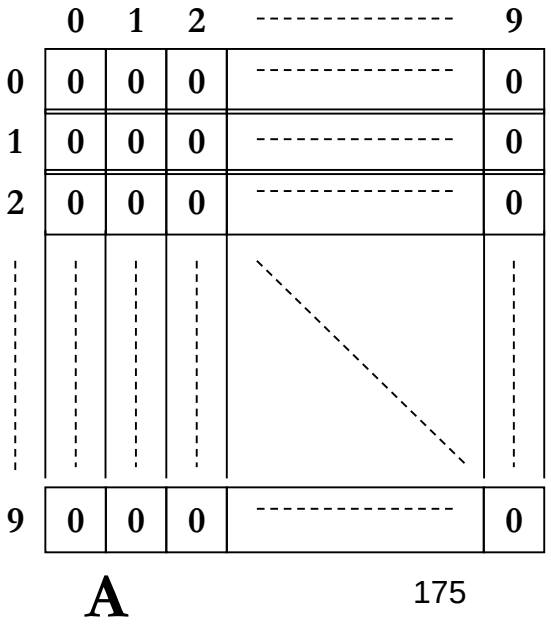
Para i = 1

j varia de 0 a 9, escrevendo na mesma linha os valores de:

A[1][0], A[1][1], ... , A[1][9]

Depois o cursor do vídeo vai para a linha seguinte

Copiar, salvar e executar



Programa 7.5: Matriz 10 x 10 nula

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10] = {0}, i, j;
   printf ("Matriz 10x10 nula:\n\n");
   system ("pause");
   printf ("\n\n");
   for (i = 0; i <= 9; i++) {
        for (j = 0; j <= 9; j++)
              printf ("%4d", A[i][j]);
        printf ("\n");
   }
   printf ("\n\n");
   system ("pause"); return 0;
}
```

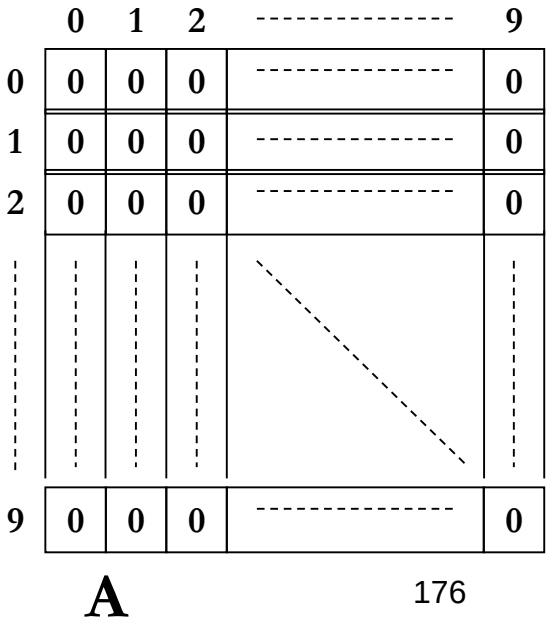
Para i = 2

j varia de 0 a 9, escrevendo na mesma linha os valores de:

A[2][0], A[2][1], ... , A[2][9]

Depois o cursor do vídeo vai para a linha seguinte

Copiar, salvar e executar



Programa 7.5: Matriz 10 x 10 nula

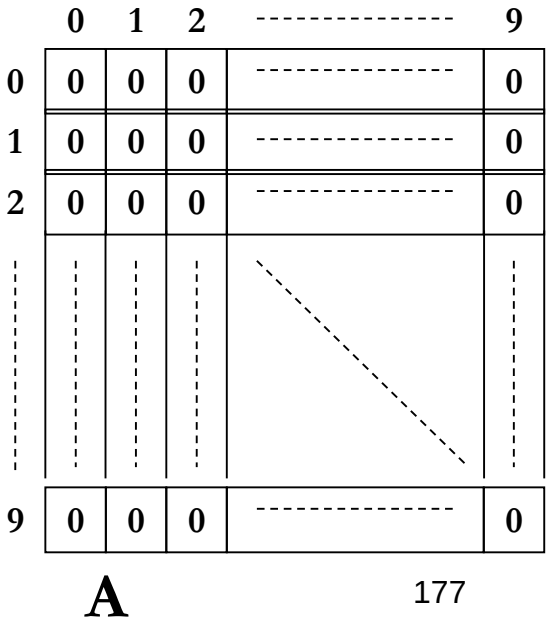
```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10] = {0}, i, j;
   printf ("Matriz 10x10 nula:\n\n");
   system ("pause");
   printf ("\n\n");
   for (i = 0; i <= 9; i++) {
        for (j = 0; j <= 9; j++)
              printf ("%4d", A[i][j]);
        printf ("\n");
   }
   printf ("\n\n");
   system ("pause"); return 0;
}
```

Copiar, salvar e executar

E assim por diante, para i = 9

j varia de 0 a 9, escrevendo na mesma linha os valores de:

A[9][0], A[9][1], ... , A[9][9]

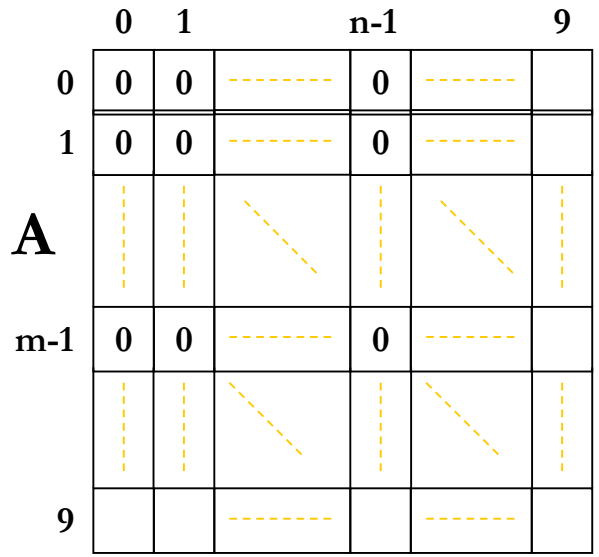


A

Programa 7.6: Matriz m x n nula

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10], i, j, m, n;
   printf ("Matriz m x n nula:\n");
   do {
        printf ("\nDigite m e n (<= 10): ");
        scanf ("%d%d", &m, &n);
   } while (m < 1 || m > 10 || n < 1 || n > 10);
   for (i = 0; i <= m-1; i++)
        for (j = 0; j <= n-1; j++)
               A[i][j] = 0;
   printf ("\n\n");
   for (i = 0; i <= m-1; i++) {
        for (j = 0; j <= n-1; j++)
               printf ("%4d", A[i][j]);
        printf ("\n");
   }
   printf ("\n\n"); system ("pause"); return 0;
}
```

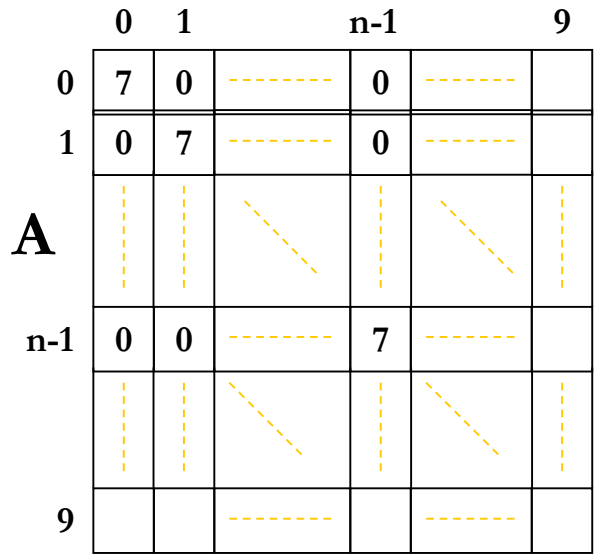
Copiar, salvar e executar



Copiar, salvar e executar

Programa 7.7: Matriz n x n diagonal

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10], i, j, n;
   printf ("Matriz n x n diagonal:\n");
   do {
        printf ("\nDigite n (<= 10): "); scanf ("%d", &n);
   } while (n < 1 || n > 10);
   for (i = 0; i <= n-1; i++)
        for (j = 0; j <= n-1; j++)
              A[i][j] = 0;
   for (i = 0; i <= n-1; i++) A[i][i] = 7;
   printf ("\n\n");
   for (i = 0; i <= n-1; i++) {
        for (j = 0; j <= n-1; j++)
              printf ("%4d", A[i][j]);
        printf ("\n\n");
   }
   printf ("\n\n"); system ("pause"); return 0;
}
```



Exercício 7.1: Colocar o número 27 na diagonal, na primeira e última linha e na primeira e última coluna de uma matriz n x n

- Exemplo:

```
Matriz n x n moldura:

Digite n (<= 10): 7

  27  27  27  27  27  27  27

  27  27   0   0   0   0  27

  27   0  27   0   0   0  27

  27   0   0  27   0   0  27

  27   0   0   0  27   0  27

  27   0   0   0   0  27  27

  27  27  27  27  27  27  27
```

Exercício 7.2: Colocar o número 82 da diagonal principal para baixo, obtendo uma matriz triangular inferior

- **Exemplo:**

```
Matriz n x n triangular inferior:

Digite n (<= 10): 6

  82   0   0   0   0   0

  82  82   0   0   0   0

  82  82  82   0   0   0

  82  82  82  82   0   0

  82  82  82  82  82   0

  82  82  82  82  82  82
```

Exercício 7.3: Obter um triangulo de Pascal

Livro 02

- **Exemplos:**

Livro 02

```
Triangulo de Pascal:

Dimensao (<= 10): 7

   1
   1   1
   1   2   1
   1   3   3   1
   1   4   6   4   1
   1   5  10  10   5   1
   1   6  15  20  15   6   1
```

Os elementos da 1ª coluna e da diagonal principal são iguais a 1

```
Triangulo de Pascal:

Dimensao (<= 10): 8

   1
   1   1
   1   2   1
   1   3   3   1
   1   4   6   4   1
   1   5  10  10   5   1
   1   6  15  20  15   6   1
   1   7  21  35  35  21   7   1
```

Os elementos acima da diagonal principal não são inicializados nem escritos

Os elementos abaixo da diagonal principal, exceto os da 1ª coluna, são dados por:

$$A[i][j] = A[i-1][j-1] + A[i-1][j]$$

Programa 7.8: Leitura e escrita de matriz

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10], m, n, i, j;
   printf ("Leitura e escrita de matriz\n\n");
   do {
        printf ("\nDimensoes m e n (<= 10): ");
        scanf ("%d%d", &m, &n);
   } while (m < 1 || m > 10 || n < 1 || n > 10);
   printf ("\n\n");
   for (i = 0; i <= m-1; i++) {
        printf ("%d elementos da linha %d: ", n, i);
        for (j = 0; j <= n-1; j++) scanf ("%d", &A[i][j]);
   }
   printf ("\n\nMatriz A:\n\n");
   for (i = 0; i <= m-1; i++) {
        for (j = 0; j <= n-1; j++) printf ("%6d", A[i][j]);
        printf ("\n\n");
   }
   printf ("\n\n"); system ("pause"); return 0;
}
```

Copiar, salvar e executar

Programa 7.9: Leitura de matriz por arquivo

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10], m, n, i, j;
   FILE *FileIn;
   FileIn = fopen ("Matriz", "r");
   printf ("Informacoes sobre matriz:");
   printf ("\n\n\tMatriz de entrada no arquivo Matriz");
   fscanf (FileIn, "%d%d", &m, &n);
   for (i = 0; i <= m-1; i++)
        for (j = 0; j <= n-1; j++)
              fscanf (FileIn, "%d", &A[i][j]);
   printf ("\n\nMatriz A:\n\n");
   for (i = 0; i <= m-1; i++) {
        for (j = 0; j <= n-1; j++)
              printf ("%4d", A[i][j]);
        printf ("\n\n");
   }
   printf ("\n\n"); system ("pause"); return 0;
}
```

Criar arquivo "Matriz" com:

```
3   4
23   65   -2   0
45   13   390   -54
9   -1   -23   7
```

Obs.: A entrada de dados não precisa ser amigável

Copiar, salvar e executar

Programa 7.10: Leitura e escrita de matriz por arquivo

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int A[10][10], m, n, i, j;
   FILE *FileIn, *FileOut;
   FileIn = fopen ("Matriz", "r");
   FileOut = fopen ("MatrizSaida", "w");
   printf ("Informacoes sobre matriz:");
   printf ("\n\n\tMatriz de entrada no arquivo Matriz");
   printf ("\n\n\tMatriz de saida no arquivo MatrizSaida");
   fscanf (FileIn, "%d%d", &m, &n);
   for (i = 0; i <= m-1; i++)
        for (j = 0; j <= n-1; j++)
              fscanf (FileIn, "%d", &A[i][j]);
   fprintf (FileOut, "Matriz A:\n\n");
   for (i = 0; i <= m-1; i++) {
        for (j = 0; j <= n-1; j++)
              fprintf (FileOut, "%4d", A[i][j]);
        fprintf (FileOut, "\n\n");
   }
   printf ("\n\n"); system ("pause"); return 0;
}
```

Copiar, salvar e executar

Abrir e examinar o arquivo "MatrizSaida"

```
Matriz A:

  23   65   -2    0

  45   13  390  -54

   9   -1  -23    7
```

7º Exercício Laboratorial: a) Aumentar o Programa 7.10 para montar e escrever, no arquivo de saída, um vetor de números inteiros contendo a quantidade de números maiores que zero de cada coluna da matriz A

- **Exemplo:** seja o seguinte conteúdo do arquivo de entrada com uma matriz 5x6:

```
  5    6
-17   21    2   -8  -57    0
 -1   64 -272  791    7   -4
-100  14   17   33  -10  320
  0    2   55   87    0   15
 81   21    2    0   27    0
```

- Saída para a matriz de entrada:

```
Matriz A:

                                                  Elementos
-------------------------------------------------------------
Linh 0:      -17       21        2       -8      -57         0
Linh 1:       -1       64     -272      791        7        -4
Linh 2:     -100       14       17       33      -10       320
Linh 3:        0        2       55       87        0        15
Linh 4:       81       21        2        0       27         0
-------------------------------------------------------------
> zero:        1        5        4        3        2         2
```

7º Exercício Laboratorial: b) Aumentar o programa do item a para montar e escrever, no arquivo de saída, um vetor de números reais contendo a média aritmética dos quadrados dos elementos de cada linha da matriz A

- **Exemplo:** Saída para a matriz de entrada:

```
Matriz A:

                                                  Elementos |   MedQuad
------------------------------------------------------------|----------
Linh 0:      -17       21        2       -8      -57        0 |     674.50
Linh 1:       -1       64     -272      791        7       -4 |  117304.45
Linh 2:     -100       14       17       33      -10      320 |   19012.33
Linh 3:        0        2       55       87        0       15 |    1802.50
Linh 4:       81       21        2        0       27        0 |    1289.17
------------------------------------------------------------|----------
> zero:        1        5        4        3        2        2
```

7º Exercício Laboratorial: c) : Aumentar o programa do item b para montar e escrever, no arquivo de saída, um vetor de números inteiros contendo o menor, em valores relativos, dos elementos de cada coluna da matriz A

- **Exemplo:** Saída para a matriz de entrada:

```
Matriz A:

                                                  Elementos |   MedQuad
------------------------------------------------------------|----------
Linh 0:      -17       21        2       -8      -57        0 |     674.50
Linh 1:       -1       64     -272      791        7       -4 |  117304.45
Linh 2:     -100       14       17       33      -10      320 |   19012.33
Linh 3:        0        2       55       87        0       15 |    1802.50
Linh 4:       81       21        2        0       27        0 |    1289.17
------------------------------------------------------------|----------
> zero:        1        5        4        3        2        2
Menor :     -100        2     -272       -8      -57       -4
```

7º Exercício Laboratorial: d) Aumentar o programa do item c para montar e escrever, no arquivo de saída, um vetor de números inteiros contendo a quantidade de elementos ímpares de cada linha da matriz A

- **Exemplo:** Saída para a matriz de entrada:

Matriz A:

						Elementos	MedQuad	Impares
Linh 0:	-17	21	2	-8	-57	0	674.50	3
Linh 1:	-1	64	-272	791	7	-4	117304.45	3
Linh 2:	-100	14	17	33	-10	320	19012.33	2
Linh 3:	0	2	55	87	0	15	1802.50	3
Linh 4:	81	21	2	0	27	0	1289.17	3
> zero:	1	5	4	3	2	2		
Menor :	-100	2	-272	-8	-57	-4		

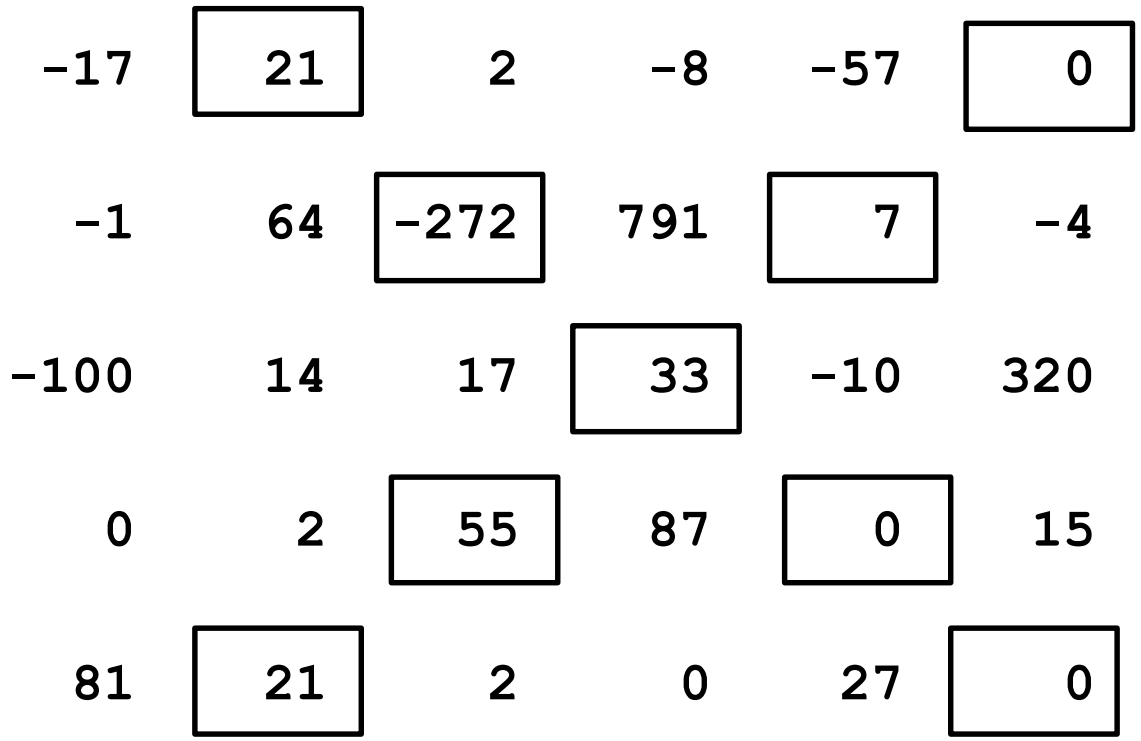
7º Exercício Laboratorial: e) Aumentar o programa do item d para montar e escrever , no arquivo de saída, outra matriz B(m x n) de inteiros e de dimensões iguais às da matriz A(m x n), tal que:

- Cada elemento **B[i][j]** (**0 ≤ i < m** e **0 ≤ j < n**) deva ser a soma dos elementos das duas diagonais da matriz **A** que contêm o elemento **A[i][j]**

- Nessa soma, o elemento **A[i][j]** deve ser contado apenas uma vez.

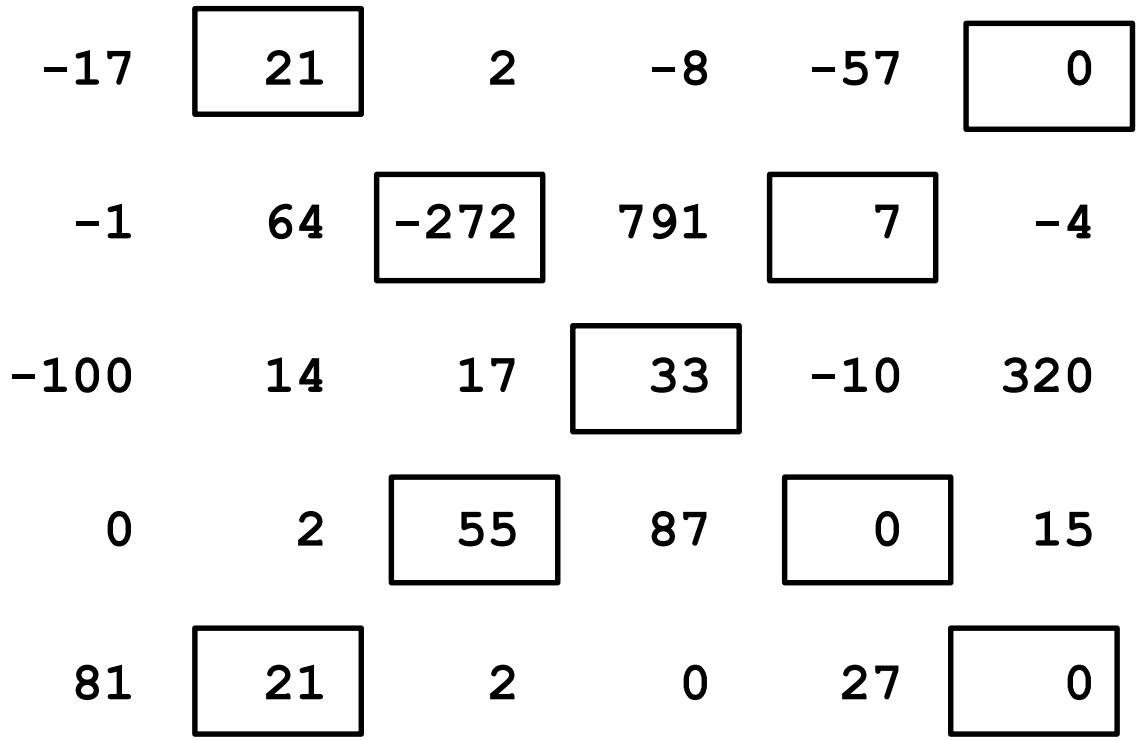
- Ver o exemplo a seguir.

- **Exemplo:** seja a seguinte matriz **A(5x6)** de entrada:



- Seja o elemento **A[2][3]**
- Sejam os elementos das duas diagonais que passam por ele

- **Exemplo:** seja a seguinte matriz **A(5x6)** de entrada:



- **B[2][3] = 21 + 0 - 272 + 7 + 33 + 55 + 0 + 21 + 0 = -135**

- Deve-se fazer este cálculo para todos os elementos de **B**

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo VIII

Cadeias de Caracteres e Estruturas

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
```

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
```

Cad1

?	?	?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
```

Cad1

?	?	?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
```

Cad1

?	?	?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Supor a digitação de 1234<enter>

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Supor a digitação de 1234<enter>

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
```

Cad1

'1'	'2'	'3'	'4'	'\0'	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

'a'	'b'	'c'	'd'	'e'	'\0'
0	1	2	3	4	5

Será escrito:

```
Cad1 = 1234
Cad2 = abcde
```

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
   printf ("\n");
   for (i = 0; i <= 9; i++)
       printf ("\n\tCad1[%d] = '%c' ou %d", i, Cad1[i], Cad1[i]);
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
   printf ("\n");
   for (i = 0; i <= 9; i++)
       printf ("\n\tCad1[%d] = '%c' ou %d", i, Cad1[i], Cad1[i]);
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

```
Cad1[0] = '1' ou 49
Cad1[1] = '2' ou 50
Cad1[2] = '3' ou 51
Cad1[3] = '4' ou 52
Cad1[4] = ' ' ou 0
Cad1[5] = 'ý' ou -20
Cad1[6] = '@' ou 64
Cad1[7] = ' ' ou 0
Cad1[8] = '☺' ou 1
Cad1[9] = ' ' ou 0
```

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
   printf ("\n");
   for (i = 0; i <= 9; i++)
       printf ("\n\tCad1[%d] = '%c' ou %d", i, Cad1[i], Cad1[i]);
   printf ("\n");
   for (i = 0; i <= 5; i++)
       printf ("\n\tCad2[%d] = '%c' ou %d", i, Cad2[i], Cad2[i]);
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
   printf ("\n");
   for (i = 0; i <= 9; i++)
       printf ("\n\tCad1[%d] = '%c' ou %d", i, Cad1[i], Cad1[i]);
   printf ("\n");
   for (i = 0; i <= 5; i++)
       printf ("\n\tCad2[%d] = '%c' ou %d", i, Cad2[i], Cad2[i]);
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

```
Cad2[0] = 'a' ou 97
Cad2[1] = 'b' ou 98
Cad2[2] = 'c' ou 99
Cad2[3] = 'd' ou 100
Cad2[4] = 'e' ou 101
Cad2[5] = ' ' ou 0
```

Programa 8.1: Leitura e escrita de cadeias de caracteres

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char Cad1[10], Cad2[] = "abcde"; int i;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad1);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
   printf ("\n");
   for (i = 0; i <= 9; i++)
       printf ("\n\tCad1[%d] = '%c' ou %d", i, Cad1[i], Cad1[i]);
   printf ("\n");
   for (i = 0; i <= 5; i++)
       printf ("\n\tCad2[%d] = '%c' ou %d", i, Cad2[i], Cad2[i]);
   printf ("\n\n"); system ("pause");
   return 0;
}
```

Cad1

‘1’	‘2’	‘3’	‘4’	‘\0’	?	?	?	?	?
0	1	2	3	4	5	6	7	8	9

Cad2

‘a’	‘b’	‘c’	‘d’	‘e’	‘\0’
0	1	2	3	4	5

Cuidado para não digitar mais do que 9 caracteres para Cad1

Copiar, salvar e executar

Programa 8.2: O formato %s para leitura de cadeias de caracteres

Copiar, salvar e executar

```
#include <stdio.h>
#include <stdlib.h>
typedef char cadeia[20];
int main () {
   cadeia Cad1, Cad2;
   printf ("Digite uma frase:\n\n");
   scanf ("%s%s", Cad1, Cad2);
   printf ("\nCad1 = %s", Cad1);
   printf ("\nCad2 = %s", Cad2);
   printf ("\n\n"); system ("pause");
   return 0;
}
```

```
Digite uma frase:

gosto muito de CES-10

Cad1 = gosto
Cad2 = muito

Digite algo para encerrar
```

Resultado de uma execução

Com o formato %s, scanf encerra a leitura para uma variável, quando encontra um espaço em branco ou um 'enter'

Programa 8.3: Comprimento de cadeias de caracteres

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef char cadeia[50];
int main () {
   cadeia Cad;
   printf ("Digite uma cadeia de caracteres:\n\n");
   scanf ("%s", Cad);
   printf ("\n\tComprimento (%s) = %d", Cad, strlen(Cad));
   printf ("\n\n"); system ("pause");
   return 0;
}
```

Copiar, salvar e executar

'strlen' retorna o nº de caracteres de uma cadeia, sem contar o '\0'

Resultado de uma execução

```
Digite uma cadeia de caracteres:

anticonstitucionalissimamente

        Comprimento (anticonstitucionalissimamente) = 29

Digite algo para encerrar
```

Programa 8.4: Conversão de binário para decimal

- Método de conversão: seja o binário **1001011** a ser convertido
- Inicialmente, o número convertido é **zerado**

Digito	Cálculo	Valor decimal
		0 (valor inicial)
1	0 x 2 + 1 =	1
0	1 x 2 + 0 =	2
0	2 x 2 + 0 =	4
1	4 x 2 + 1 =	9
0	9 x 2 + 0 =	18
1	18 x 2 + 1 =	37
1	37 x 2 + 1 =	75 (Valor final)

Programa 8.4: Conversão de binário para decimal

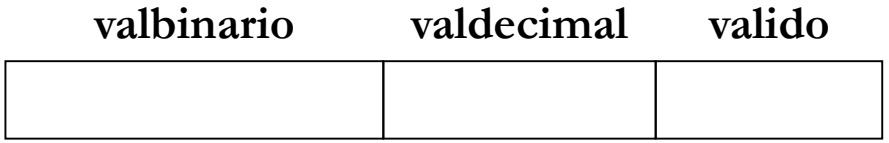
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*Declaracoes do tipo logic e do tipo cadeia de caracteres*/

typedef char logic;
const logic True = 1, False = 0;
typedef char cadeia[30];

/*Declaracao da struct binario*/

typedef struct binario binario;
struct binario {
    cadeia valbinario; int valdecimal;
    logic valido;
};
```



struct binario

Programa 8.4: Conversão de binário para decimal

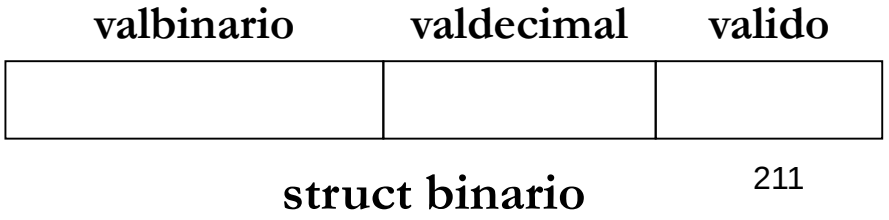
```
/* Funcao main, declaracoes, leitura do binario e inicializacoes */

int main () {
   binario bin; int i, n;
   printf ("Digite um numero binario: "); scanf ("%s", bin.valbinario);
   bin.valido = True; n = strlen (bin.valbinario);

/* Verificacao da correcao do numero binario*/

   for (i = 0; i < n && bin.valido == True; i++)
         if (bin.valbinario[i] != '0' && bin.valbinario[i] != '1')
             bin.valido = False;

   if (bin.valido == True) {
```



Programa 8.4: Conversão de binário para decimal

```
/* Calculo do valor em decimal de um binario correto */

        for (i = 0, bin.valdecimal = 0; i < n; i++)
             bin.valdecimal = 2*bin.valdecimal + (bin.valbinario[i] - '0');
        printf ("\n\t(%s)2 = (%d)10", bin.valbinario, bin.valdecimal);
   }

/* Aviso de numero binario incorreto*/

   else
        printf ("\n\t%s nao representa um binario", bin.valbinario);

/* Encerramento do programa */

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

Copiar, salvar e executar

valbinario	valdecimal	valido

Programa 8.4: Conversão de binário para decimal

```
/* Calculo do valor em decimal de um binario correto */

        for (i = 0, bin.valdecimal = 0; i < n; i++)
             bin.valdecimal = 2*bin.valdecimal + (bin.valbinario[i] - '0');
        printf ("\n\t(%s)2 = (%d)10", bin.valbinario, bin.valdecimal);
   }

/* Aviso de numero binario incorreto*/

   else
        printf ("\n\t%s nao representa um binario", bin.valbinario);

/* Encerramento do programa */

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

Copiar, salvar e executar

```
Digite um numero binario: 1001011

        (1001011)2 = (75)10

Digite algo para encerrar:
```

1º exemplo de resultado

Programa 8.4: Conversão de binário para decimal

```
/* Calculo do valor em decimal de um binario correto */

        for (i = 0, bin.valdecimal = 0; i < n; i++)
             bin.valdecimal = 2*bin.valdecimal + (bin.valbinario[i] - '0');
        printf ("\n\t(%s)2 = (%d)10", bin.valbinario, bin.valdecimal);
   }

/* Aviso de numero binario incorreto*/

   else
        printf ("\n\t%s nao representa um binario", bin.valbinario);

/* Encerramento do programa */

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

Copiar, salvar e executar

```
Digite um numero binario: 100711

        100711 nao eh um binario

Digite algo para encerrar:
```

2º exemplo de resultado

Exercício 8.1: Conversão de bases de 2 a 9 para decimal

- Método de conversão: seja o número (**7504)8** a ser convertido para decimal

- Inicialmente, o número convertido é **zerado**

Digito	Cálculo	Valor decimal
		0 (valor inicial)
7	0 x 8 + 7 =	7
5	7 x 8 + 5 =	61
0	61 x 8 + 0 =	488
4	488 x 8 + 4 =	3908 (Valor final)

Exercício 8.1: Conversão de bases de 2 a 9 para decimal

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

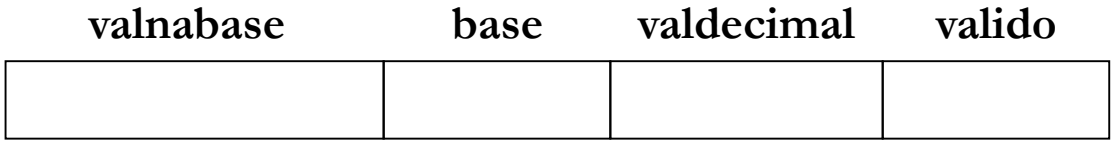
/*Declaracoes do tipo logic e do tipo cadeia de caracteres*/

typedef char logic;
const logic True = 1, False = 0;
typedef char cadeia[30];

/*Declaracao da struct numero*/

typedef struct numero numero;
struct numero {
    cadeia valnabase; int base, valdecimal;
    logic valido;
};
```

Completar, salvar e executar o seguinte programa à esquerda



struct numero

Exercício 8.1: Conversão de bases de 2 a 9 para decimal

```
/*Funcao main, declaracoes e leitura do numero e da base*/

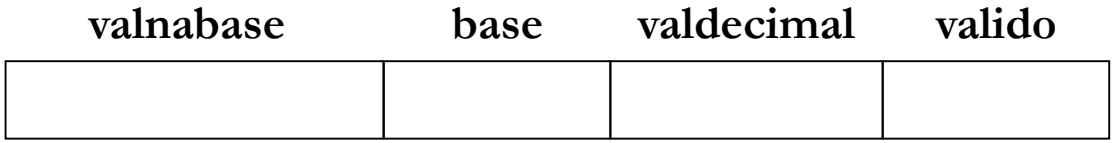
int main () {
   numero num; int i, n;
   - - - - - - - - - - - - - - - - - -

/* Verificacao da correcao da base e inicializacoes */

   - - - - - - - - - - - - - - - - - -

/* Verificacao da correcao do numero na base*/

   - - - - - - - - - - - - - - - - - -
```



struct numero

Exercício 8.1: Conversão de bases de 2 a 9 para decimal

```
/* Calculo do valor em decimal de um numero correto */

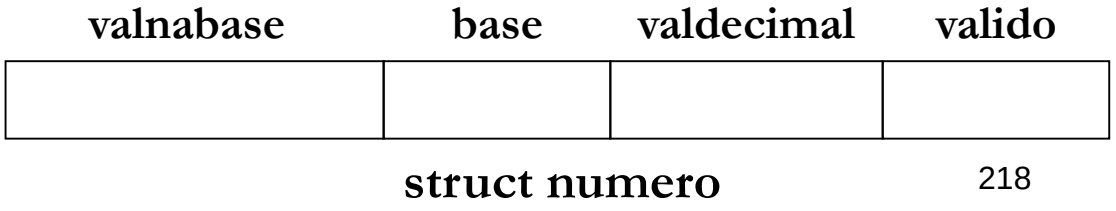
    - - - - - - - - - - - - - - - - - - -

/* Aviso de numero incorreto*/

    - - - - - - - - - - - - - - - - - - -

/* Encerramento do programa */

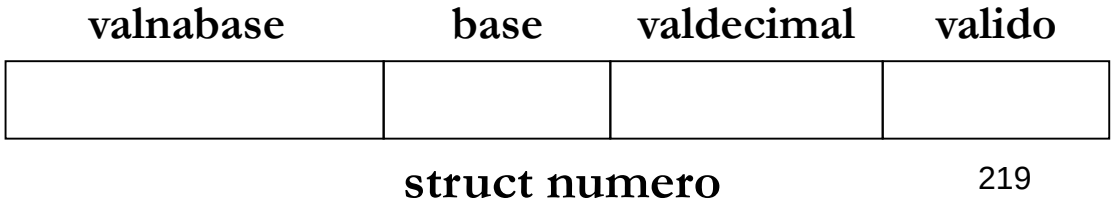
    - - - - - - - - - - - - - - - - - - -;
}
```



Exercício 8.2: Conversão de bases de 2 a 20 para decimal

- Fazer um **programa semelhante** ao do exercício anterior, **ampliando** a abrangência das bases para **[2, 20]**
- Para bases **acima de 10**, aceitar apenas **letras maiúsculas**
- Usar a mesma **struct numero** do exercício anterior

```
typedef struct numero numero;
struct numero {
    cadeia valnabase; int base, valdecimal;
    logic valido;
};
```



8º Exercício Laboratorial: Somatória dos valores decimais de números e bases lidos

- Ampliar o programa do exercício anterior para ler de um arquivo **vários números e suas bases**, e calcular a **somatória em decimal** dos números e bases que estiverem **corretos**

- O número de pares **[numero-base]** deve ser lido do mesmo arquivo, **antes** da leitura desses pares

- Antes da soma, cada par **[número-base]** lido deve ser armazenado num **vetor de structs numero**, mesmo se incorreto

- Esse **vetor** deve ser **um dos campos** de outra **struct**; o **outro campo** deve ser o **número de structs** do vetor

8º Exercício Laboratorial: Somatória dos valores decimais de números e bases lidos

- Cada par **[numero, base]** lido deve ser:
  - **Testado**
  - **Convertido** para decimal, se estiver correto
  - **Armazenado** no vetor de structs
- Depois, deve ser realizada a **somatória** dos valores decimais dos **números corretos** do vetor
- No final, devem ser escritos num **outro arquivo** todas as conversões, bem sucedidas ou não, registradas no **vetor de structs numero** e a **somatória**

Exemplo:

Diagrama da **struct** com o vetor de **structs numero**:

TabNum

ntab | 8

VetNum

valnabase	base	valdecimal	valido
101101101	2	365	V
423014	5	14134	V
42501	5	-1	F
AB3K	12	-1	F
AB375	12	226889	V
FFFF	16	65535	V
FFGF	16	-1	F
JJJJ	20	159999	V

Seja o seguinte conteúdo do arquivo de entrada de dados:

```
8
101101101    2
423014       5
42501        5
AB3K         12
AB375        12
FFFF         16
FFGF         16
JJJJ         20
```

Exemplo:

Seja o seguinte conteúdo do arquivo de entrada de dados:

```
8
101101101    2
423014       5
42501        5
AB3K         12
AB375        12
FFFF         16
FFGF         16
JJJJ         20
```

Exemplo:

- Possíveis **resultados** no arquivo de saída:

```
C O N V E R S O E S   D E   B A S E S   P A R A   A   B A S E   10

Foram realizadas 8 conversoes:

(101101101)2 = (365)10
(423014)5 = (14134)10
(42501)5 = Dados incompativeis
(AB3K)12 = Dados incompativeis
(AB375)12 = (226889)10
(FFFF)16 = (65535)10
(FFGF)16 = Dados incompativeis
(JJJJ)20 = (159999)10

Somatoria dos numeros validos = (466922)10
```

Seja o seguinte conteúdo do arquivo de entrada de dados:

```
8
101101101    2
423014       5
42501        5
AB3K         12
AB375        12
FFFF         16
FFGF         16
JJJJ         20
```

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo IX

Estruturas e Subprogramas sem Parâmetros

Programa 9.1: Montagem da tabela de times de um campeonato de futebol de pontos corridos com 2 turnos

```
typedef char cadeia[21];

typedef struct time time;
struct time {
   cadeia nome; int pontos, vitorias, golspro, golscontra,
   saldogols;
};
```

Sejam as declarações à esquerda

tipo time



Livro 02

nome | pontos | vitorias | golspro | golscontra | saldogols

```
typedef struct tabelatimes tabelatimes;
struct tabelatimes {
   time VetorTimes[30]; int ntimes;
};

tabelatimes TabTimes;
```

Elas implementam a estrutura a seguir

VetorTimes

	nome	pontos	vitorias	golspro	golscontra	saldogols
0						
1						
2						
3						
. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .
27						
28						
29						

ntimes [ ]

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

/* Declarações sobre um time e sobre uma tabela de times */

typedef char cadeia[31];
typedef struct time time;
struct time {
   cadeia nome;
   int pontos, vitorias, golspro, golscontra, saldogols;
};

typedef struct tabelatimes tabelatimes;
struct tabelatimes {
   time VetorTimes[30];
   int ntimes;
};
```

Copiar, salvar e executar o programa à esquerda

Continua no próximo slide

```
/* Variaveis globais */

tabelatimes TabTimes;
FILE *filein, *fileout;

/* Prototipos das funcoes auxiliares */

void InicializarArquivos (void);
void LerArmazenarTimes (void);
void EscreverTabTimes (void);

/* Funcao main: Abre e inicializa os arquivos de entrada e saida,
   le os times, armazenando-os, e escreve o conteudo da tabela de times */

int main () {
   InicializarArquivos ();
   LerArmazenarTimes ();
   EscreverTabTimes ();
   printf ("\n\n"); system ("pause"); return 0;
}
```

Protótipo de função: declaração da função antes de sua definição; possibilita a definição da função depois da função main

Continua no próximo slide

```
/* Funcao InicializarArquivos: escreve avisos no video, abre os
    arquivos de entrada e saida e escreve titulo no arquivo de
    saida */

void InicializarArquivos () {
   printf ("RESULTADOS DE UM CAMPEONATO\n\n");
   printf ("Entrada de dados no arquivo Campeonato.dat \n\n");
   printf ("Relatorio dos Resultados no arquivo Campeonato \n\n");
   filein = fopen (" Campeonato.dat ", "r");
   fileout = fopen ("Campeonato", "w");
   fprintf (fileout,
       "R E S U L T A D O S   D E   U M   C A M P E O N A T O\n\n");
}
```

Todos os dados de entrada para o programa deverão ser lidos do arquivo Campeonato.dat

A tabela dos times e as outras estruturas que virão depois serão escritas no arquivo Campeonato

Como a entrada e a saída serão feitas por arquivos, o programa não precisa ser amigável

Continua no próximo slide

```
/* Funcao LerArmazenarTimes: le os times do campeonato e
    os armazena na tabela de times, inicializando os dados
    sobre eles */

void LerArmazenarTimes () {
   int i, len; char lixo;
   fscanf (filein, "%d%c", &TabTimes.ntimes, &lixo);
   for (i = 0; i <= TabTimes.ntimes-1; i++) {
       fgets (TabTimes.VetorTimes[i].nome, 25, filein);
       len = strlen(TabTimes.VetorTimes[i].nome);
       TabTimes.VetorTimes[i].nome[len-1] = '\0';
       TabTimes.VetorTimes[i].pontos =
       TabTimes.VetorTimes[i].vitorias =
       TabTimes.VetorTimes[i].golspro =
       TabTimes.VetorTimes[i].golscontra =
       TabTimes.VetorTimes[i].saldogols = 0;
   }
}
```

Leitura do <\n> depois do 5

Eliminação do <\n> antes do <\0>

Exemplo de conteúdo do arquivo de entrada de dados:

```
5<\n>
Alemanha<\n>
Brasil<\n>
Estados Unidos<\n>
Gra Bretanha<\n>
Argentina<\n>
```

Continua no próximo slide

```
/* Funcao EscreverTabTimes: escreve todo o conteudo
    da tabela de times */

void EscreverTabTimes () {
   int i;
   fprintf (fileout, "Tabela de desempenho dos times:\n\n");

/* Cabecalho da tabela */

   fprintf (fileout, "%22s | %9s | %9s | %9s | %9s | %9s\n",
       "Time", "Pontos", "Vitorias", "G Pro", "G Contra",
       "Saldo G");
   fprintf (fileout,
"-------------------------------------------------------------------------\n");
```

Continua no próximo slide

```
/* Escrita das informacoes sobre os times */

   for (i = 0; i <= TabTimes.ntimes-1; i++)
        fprintf (fileout, "%22s | %9d | %9d | %9d | %9d | %9d\n",
        TabTimes.VetorTimes[i].nome,
        TabTimes.VetorTimes[i].pontos,
        TabTimes.VetorTimes[i].vitorias,
        TabTimes.VetorTimes[i].golspro,
        TabTimes.VetorTimes[i].golscontra,
        TabTimes.VetorTimes[i].saldogols);
}
```

```
RESULTADOS DE UM CAMPEONATO

Entrada de dados no arquivo Campeonato.dat

Relatorio dos Resultados no arquivo Campeonato



Pressione . . .
```

Exemplo de resultados de uma execução

No vídeo

No arquivo de saída:

```
R E S U L T A D O S   D E   U M   C A M P E O N A T O

Tabela de desempenho dos times:

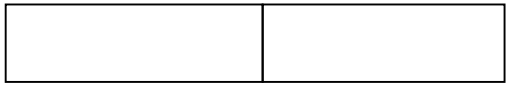
                Time |    Pontos |  Vitorias |     G Pro |  G Contra |   Saldo G
-------------------------------------------------------------------------------
             Alemanha |         0 |         0 |         0 |         0 |         0
               Brasil |         0 |         0 |         0 |         0 |         0
       Estados Unidos |         0 |         0 |         0 |         0 |         0
         Gra Bretanha |         0 |         0 |         0 |         0 |         0
            Argentina |         0 |         0 |         0 |         0 |         0
```

Livro 02

Exercício 9.1: Montagem de uma matriz com os resultados dos jogos do campeonato

Sejam as declarações à esquerda

```
typedef struct resultado resultado;
struct resultado {int local, visitante;};
```

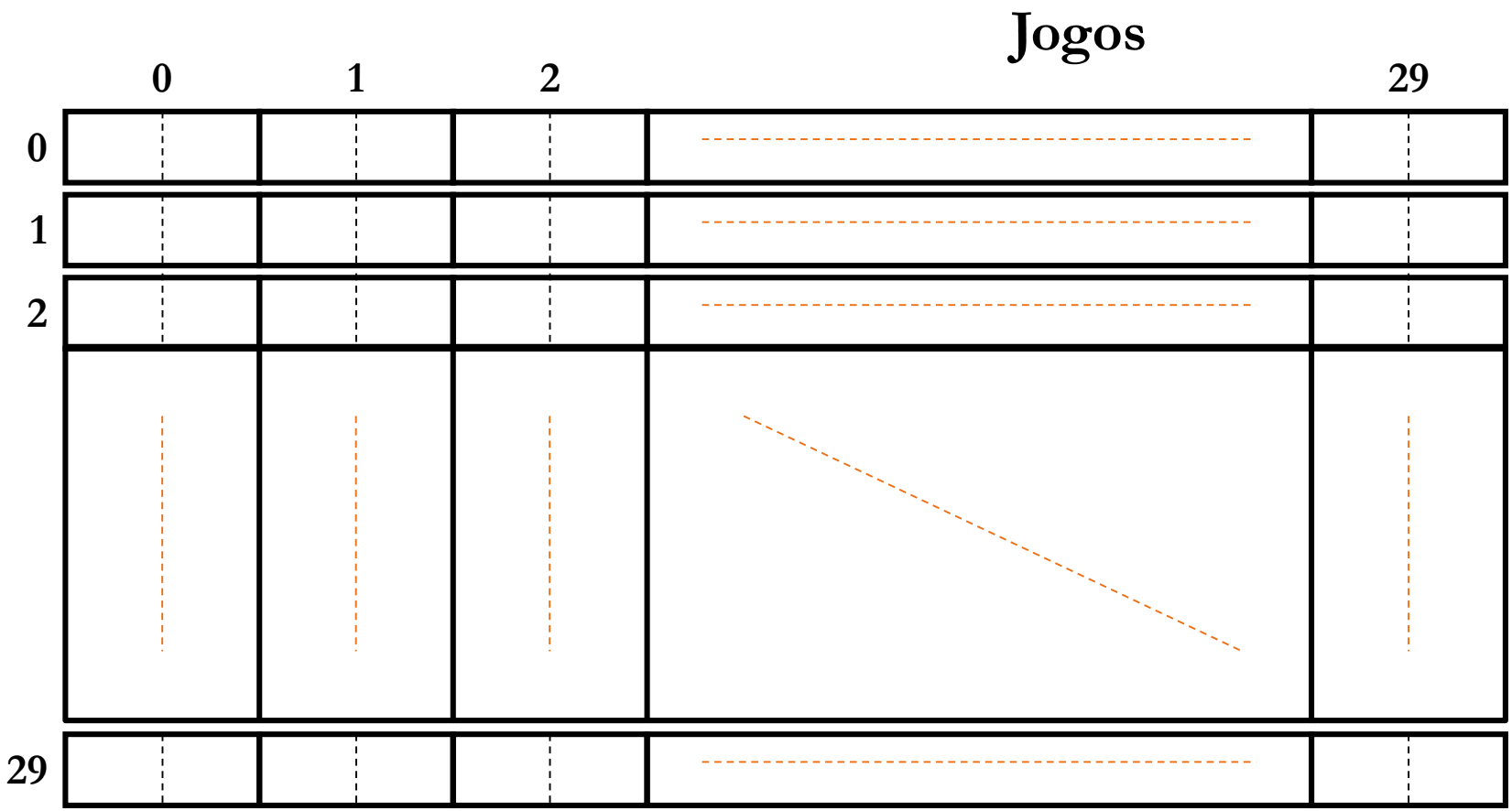


tipo resultado

local visitante

```
resultado Jogos[30][30];
```

Elas implementam a estrutura a seguir


Jogos
0
1
2
29
0
1
2
29
Livro 02

- Inserir as declarações sobre a **matriz Jogos** no programa anterior

- Incluir a **leitura por arquivo** dos resultados de todos os jogos do campeonato (1º e 2º turnos), armazenando-os na matriz **Jogos**

  - Os **resultados dos jogos** devem ser colocados organizadamente no **arquivo de entrada**

- Programar a **escrita** da matriz **Jogos** no **arquivo de saída**

- A **ordem dos jogos** no arquivo de entrada deve ser conforme o **slide** a seguir

```
Ordem dos resultados dos jogos:

3 1    Jogo do 1° time versus 2° time
4 2    Jogo do 1° time versus 3° time
2 0    Jogo do 1° time versus 4° time
1 1    Jogo do 1° time versus 5° time
3 2    Jogo do 2° time versus 1° time
2 1    Jogo do 2° time versus 3° time
2 2    .
1 2    .
2 3    .
1 2    .
3 1    .
2 1    .
3 3    .
2 1    .
4 2    .
0 0    .
2 2    .
2 1    .
1 0    Jogo do 5° time versus 3° time
1 1    Jogo do 5° time versus 4° time
```

Ordem dos resultados dos jogos:

```
3 1
4 2
2 0
1 1
3 2
2 1
2 2
1 2
2 3
1 2
3 1
2 1
3 3
2 1
4 2
0 0
2 2
2 1
1 0
1 1
```

Jogos do 1º time como mandante

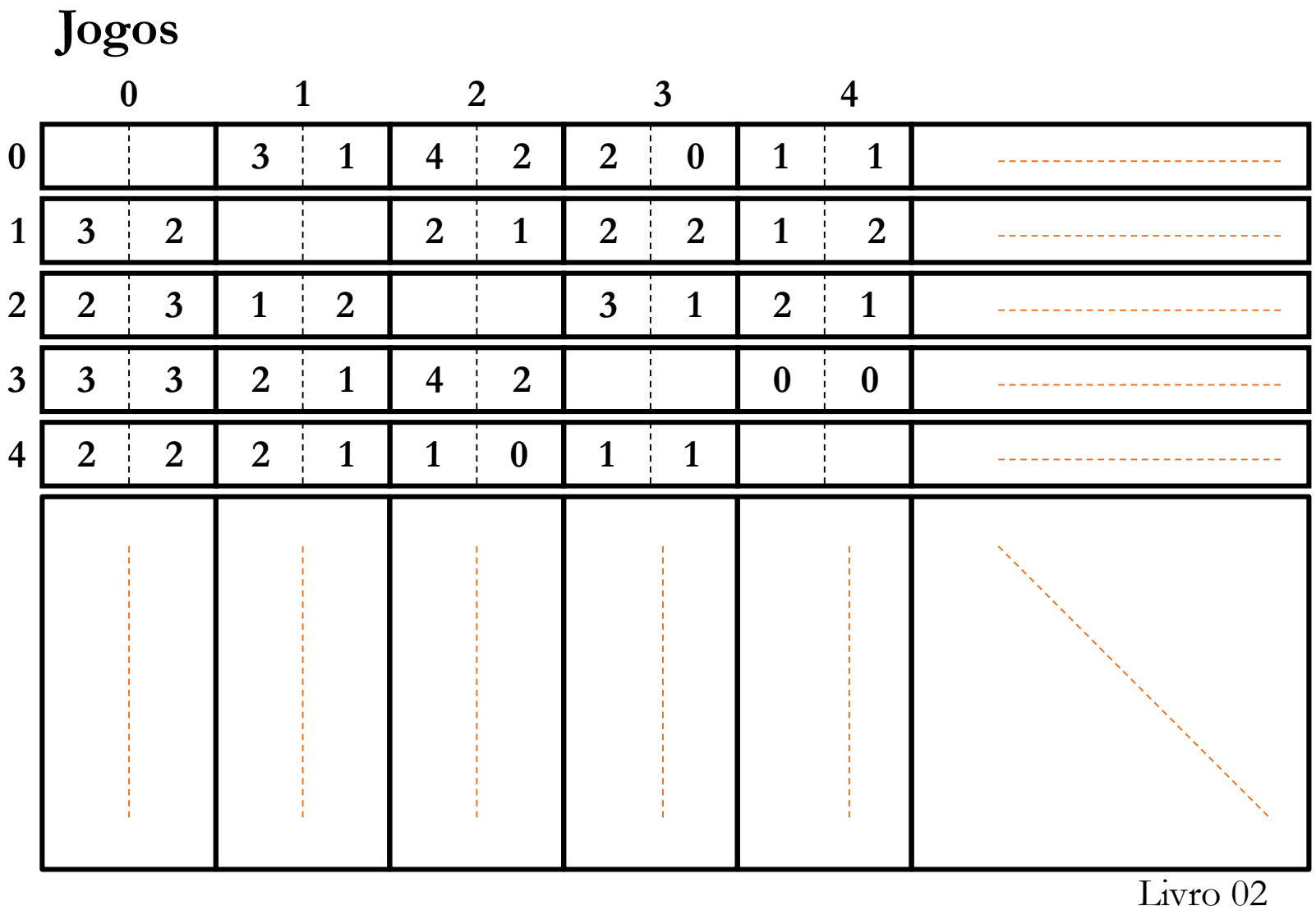
Jogos do 2º time como mandante

Jogos do 3º time como mandante

Jogos do 4º time como mandante

Jogos do 5º time como mandante

Esses resultados deverão ser armazenados na matriz Jogos, conforme o slide a seguir



- **A escrita desta matriz no arquivo poderá ser conforme o slide a seguir**

Resultados dos jogos:

	Ale	Gra	Est	Bra	Arg
Alemanha	#####	3 x 1	4 x 2	2 x 0	1 x 1
Brasil	3 x 2	#####	2 x 1	2 x 2	1 x 2
Estados Unidos	2 x 3	1 x 2	#####	3 x 1	2 x 1
Gra Bretanha	3 x 3	2 x 1	4 x 2	#####	0 x 0
Argentina	2 x 2	2 x 1	1 x 0	1 x 1	#####

Livro 02

- A função **main** deve ser alterada para conter chamada para a **leitura dos resultados** dos jogos:

```
int main () {
      InicializarArquivos ();
      LerArmazenarTimes ();
      EscreverTimes ();
      LerArmazenarJogos ();
      EscreverJogos ();
      EscreverTabTimes ();
      printf ("\n\n"); system ("pause"); return 0;
}
```

Escrever somente os times da tabela de times

Ler os resultados dos jogos, armazenando-os na matriz Jogos

Escrever a matriz Jogos

- Estas **três novas funções** devem ser **programadas**

Exercício 9.2: Preenchimento da tabela de times

Incluir no programa anterior:

- Contabilização do número de **pontos** de cada time (vitória = 3 ptos, empate = 1 pto, derrota = 0 pto)
- Contabilização do número de **vitórias**, **gols marcados**, **gols sofridos** e **saldo de gols** de cada time
- Preenchimento dos campos **pontos**, **vitorias**, **golspro**, **golscontra** e **saldogols** da **TabTimes**, conforme a contabilização acima
- Escrita do conteúdo da **TabTimes**, agora preenchida

- A função **main** deve ser alterada para conter chamada para o **preenchimento** da tabela de times:

```
int main () {
      InicializarArquivos ();
      LerArmazenarTimes ();
      EscreverTimes ();
      LerArmazenarJogos ();
      EscreverJogos ();
      PreencherTabTimes ();
      EscreverTabTimes ();
      printf ("\n\n"); system ("pause"); return 0;
}
```

Preencher a tabela de times, de acordo com a matriz Jogos

- Esta **nova função** deve ser **programada**

Exemplo de resultado no arquivo de saída

```
R E S U L T A D O S   D E   U M   C A M P E O N A T O

Listagem dos times:

                  Time
----------------------
               Alemanha
                 Brasil
         Estados Unidos
           Gra Bretanha
              Argentina

Resultados dos jogos:

                        Ale      Bra      Est      Gra      Arg
               Alemanha #####    3 x 1    4 x 2    2 x 0    1 x 1
                 Brasil 3 x 2    #####    2 x 1    2 x 2    1 x 2
         Estados Unidos 2 x 3    1 x 2    #####    3 x 1    2 x 1
           Gra Bretanha 3 x 3    2 x 1    4 x 2    #####    0 x 0
              Argentina 2 x 2    2 x 1    1 x 0    1 x 1    #####

Tabela de desempenho dos times:

                  Time |    Pontos |  Vitorias |     G Pro |  G Contra |   Saldo G
-----------------------------------------------------------------------------------
               Alemanha |        15 |         4 |        20 |        14 |         6
                 Brasil |        10 |         3 |        13 |        15 |        -2
         Estados Unidos |         6 |         2 |        13 |        18 |        -5
           Gra Bretanha |        10 |         2 |        13 |        14 |        -1
              Argentina |        13 |         3 |        10 |         8 |         2
```

Exercício 9.3: Classificação dos times, ordenação decrescente e escrita da tabela de times

- Prioridades para a classificação:

1. **Pontuação**
2. **Número de vitórias**
3. **Saldo de gols**
4. **Gols marcados**
5. **Ordem alfabética**

Para ordenar a tabela de times, pode-se adaptar o método Bubble-Sort visto em aula teórica

Mas não é proibido usar outro método

- Na escrita da **tabela ordenada**, os gols sofridos devem ser **omitidos** (fazer outra função para escrever **TabTimes**)

- A função **main** deve ser alterada para conter chamada para a **ordenação dos resultados** dos jogos:

```
int main () {
      InicializarArquivos ();
      LerArmazenarTimes ();
      EscreverTimes ();
      LerArmazenarJogos ();
      EscreverJogos ();
      PreencherTabTimes ();
      EscreverTabTimes ();
      OrdenarTabTimes ();
      EscreverClassificacaoFinal ();
      printf ("\n\n"); system ("pause"); return 0;
}
```

Ordenar a tabela de times em ordem decrescente, conforme as prioridades

Escrever tabela de times ordenada

- Estas **duas novas funções** devem ser **programadas**

A saída pode ser conforme a do slide a seguir

9º Exercício Laboratorial: Realizar os Exercícios 9.1, 9.2 e 9.3

```
R E S U L T A D O S   D E   U M   C A M P E O N A T O

Listagem dos times:

                Time
----------------------
             Alemanha
               Brasil
       Estados Unidos
         Gra Bretanha
            Argentina

Resultados dos jogos:

                      Ale     Bra     Est     Gra     Arg
            Alemanha  #####   3 x 1   4 x 2   2 x 0   1 x 1
              Brasil  3 x 2   #####   2 x 1   2 x 2   1 x 2
      Estados Unidos  2 x 3   1 x 2   #####   3 x 1   2 x 1
        Gra Bretanha  3 x 3   2 x 1   4 x 2   #####   0 x 0
           Argentina  2 x 2   2 x 1   1 x 0   1 x 1   #####

Tabela de desempenho dos times:

                Time |    Pontos |  Vitorias |     G Pro |  G Contra |   Saldo G
---------------------------------------------------------------------------------
             Alemanha |        15 |         4 |        20 |        14 |         6
               Brasil |        10 |         3 |        13 |        15 |        -2
       Estados Unidos |         6 |         2 |        13 |        18 |        -5
         Gra Bretanha |        10 |         2 |        13 |        14 |        -1
            Argentina |        13 |         3 |        10 |         8 |         2

Classificacao Final:

                Time |    Pontos |  Vitorias |   Saldo G |     G Pro
-------------------------------------------------------------------
 1)          Alemanha |        15 |         4 |         6 |        20
 2)        Argentina  |        13 |         3 |         2 |        10
 3)            Brasil |        10 |         3 |        -2 |        13
 4)      Gra Bretanha |        10 |         2 |        -1 |        13
 5)    Estados Unidos |         6 |         2 |        -5 |        13
```

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo X

Subprogramas com Parâmetros e Recursividade

Programa 10.1: Subprograma bem simples para somar

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

int soma (int x, int y)
{return x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   c = soma (a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Copiar, salvar e executar

Programa 10.2: Mesmo efeito, porém sem retornar valor

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Copiar, salvar e executar

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Função main



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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Função main



Leitura de a e b: 151 346

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Função main

a	151	b	346	c	

Leitura de a e b: 151 346

Função soma

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

void soma (int *z, int x, int y)
{*z = x + y;}
```

x | y | z

Função main

```
int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

a 151 | b 346 | c

Alocação das variáveis de soma

Função soma

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

void soma (int *z, int x, int y)
{*z = x + y;}
```

x [ ] y [ ] z [ ]

Função main

```
int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

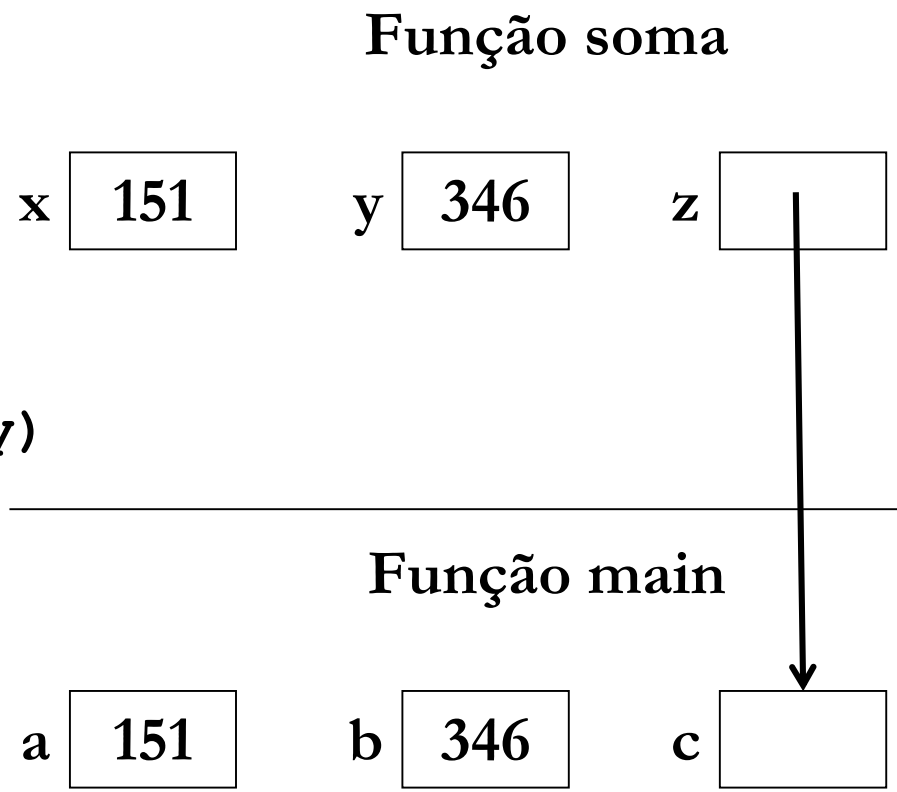
a [151] b [346] c [ ]

Passagem dos argumentos aos parâmetros

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

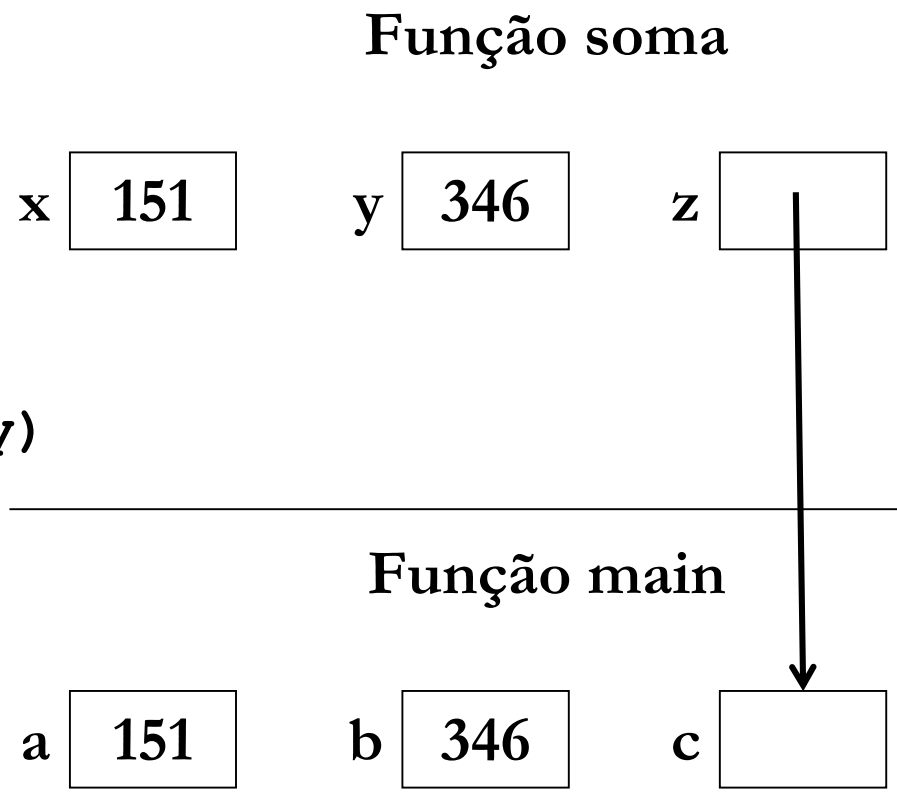


Passagem dos argumentos aos parâmetros

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```



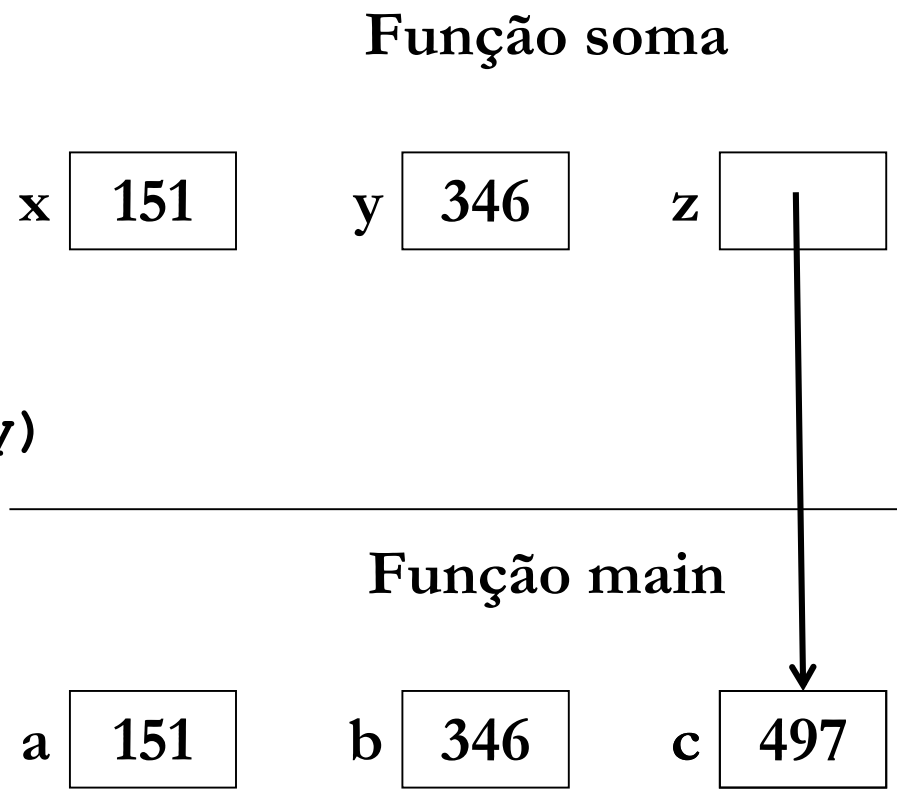
z recebeu o endereço de c

z é um ponteiro (apontando para c)

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main () {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

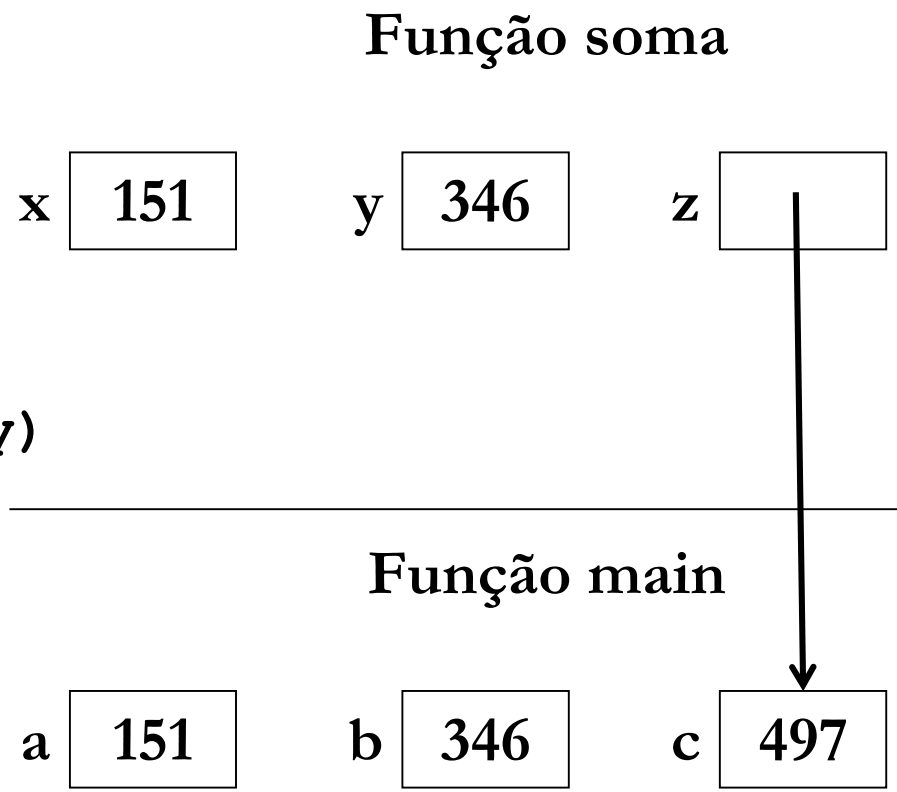


O local cujo endereço está em z (*z) recebe o valor de x+y

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```



Desalocação das variáveis de soma

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Função main

a 151 b 346 c 497

Desalocação das variáveis de soma

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

void soma (int *z, int x, int y)
{*z = x + y;}

int main ( ) {
   int a, b, c;
   printf ("Digite a e b: ");
   scanf ("%d%d", &a, &b);
   soma (&c, a, b);
   printf ("\nc = a + b = %d + %d = %d", a, b, c);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Função main

a	151	b	346	c	497

Será escrito: `c = a + b = 151 + 346 = 497`

Programa 10.3: Cálculo recursivo de fatorial

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

int fat (int n) {
   int f;
   if (n < 0) f = -1;
   else if (n <= 1) f = 1;
   else f = n * fat(n - 1);
   return f;
}

int main() {
   int n;
   printf ("Calculo de fatorial de n");
   printf ("\n\n\tDigite n: ");
   scanf ("%d", &n);
   printf ("\n\tFat(%d) = %d", n, fat(n));
   printf("\n\n"); system ("pause");
   return 0;
}
```

Formula recursiva

$$n! = \begin{cases} -1, & \text{para } n < 0 \\ 1, & \text{para } 0 \leq n \leq 1 \\ n * (n-1)!, & \text{para } n > 1 \end{cases}$$

Copiar, salvar e executar

Programa 10.4: Cálculo recursivo instrumentado de fatorial

Alterar a função fat do programa anterior

```
int fat (int n) {
   int f;
   static int v = 0; int i;
   v++;
   printf ("\n");
   for (i = 1; i <= v; i++) printf ("   ");
   printf ("Entrada em fat versao %d; n = %d; ", v, n);
   system ("pause");
   if (n < 0) f = -1;
   else if (n <= 1) f = 1;
   else f = n * fat(n - 1);
   printf ("\n");
   for (i = 1; i <= v; i++) printf ("   ");
   printf ("Saida de fat versao %d; n = %d; fat = %d; ",
       v, n, f);
   system ("pause");
   v--;
   if (v == 0) printf ("\n");
   return f;
}
```

Programa 10.4: Cálculo recursivo instrumentado de fatorial

Alterar a função fat do programa anterior

Copiar, salvar e executar

Exercício 10.1: Potência de expoentes inteiros e não negativos

A^n (n inteiro e não negativo)

- Escrever um programa para implementar a seguinte **fórmula recursiva**:

$$A^n = \begin{cases} 1, & \text{para } n = 0 \\ A * A^{n-1}, & \text{para } n > 0 \end{cases}$$

- A função **main** pode ser parecida com a do programa do **fatorial**

Exercício 10.2: Raiz quadrada

- Escrever um programa para implementar a seguinte **fórmula recursiva**:

$$\text{RaizQuad}(n, a, e) = \begin{cases} a, & \text{p/ } (|a^2 - n| < e) \\ \text{RaizQuad}(n, (a^2+n)/(2*a), e), & \text{caso contrário} \end{cases}$$

- **a** é uma **aproximação** para a raiz quadrada; **e** é a **precisão** desejada

- A **primeira aproximação** e a **precisão** devem ser lidas na função **main**

Exercício 10.3: Números de Fibonacci (procriação de coelhos)

- Gerar a sequência de números de **Fibonacci** para um dado valor de **n**, usando a seguinte **fórmula recursiva**:

$$\text{Fib}(n) = \begin{cases} -1, & p/\ n < 0 \\ 0, & p/\ n = 0 \\ 1, & p/\ n = 1 \\ \text{Fib}(n-2) + \text{Fib}(n-1), & p/\ n > 1 \end{cases}$$

- O programa deve montar uma **tabela** com os números de **Fibonacci** e com o número de chamadas recursivas para gerar cada um deles

- Para tanto, o programa deve ser devidamente **instrumentado** (ver, como exemplo, a tabela a seguir)

Exemplo: tabela para n = 20

n	Fib(n)	Chamadas recursivas
0	0	1
1	1	1
2	1	3
3	2	5
4	3	9
5	5	15
6	8	25
7	13	41
8	21	67
9	34	109
10	55	177
11	89	287
12	144	465
13	233	753
14	377	1219
15	610	1973
16	987	3193
17	1597	5167
18	2584	8361
19	4181	13529
20	6765	21891

Exercício 10.4: Cálculo de matriz menor-complementar

- **Matriz menor-complementar:** usada para o cálculo recursivo de **determinantes** pela regra de **Laplace**

- O **10º** Exercício Laboratorial será uma instrumentação do cálculo recursivo de **determinantes** por **Laplace**

- Seja uma matriz quadrada **A(n x n)**

- Matriz **menor-complementar (i, j)** de **A** é aquela obtida pela eliminação da **linha i** e da **coluna j** de **A**

- Simbolicamente, **MenorComplementar (A, i, j)**

Exemplo: seja a seguinte matriz **A(5 x 5):**

15	23	-18	87	8
-1	67	25	-62	40
7	32	-84	43	-9
87	14	-4	22	50
-33	97	25	-6	77

- **MenorComplementar (A, 3, 1):**

15	-18	87	8
-1	25	-62	40
7	-84	43	-9

-33	25	-6	77

Exemplo: seja a seguinte matriz **A(5 x 5):**

15	23	-18	87	8
-1	67	25	-62	40
7	32	-84	43	-9
87	14	-4	22	50
-33	97	25	-6	77

- **MenorComplementar (A, 3, 1):**

15	-18	87	8
-1	25	-62	40
7	-84	43	-9
-33	25	-6	77

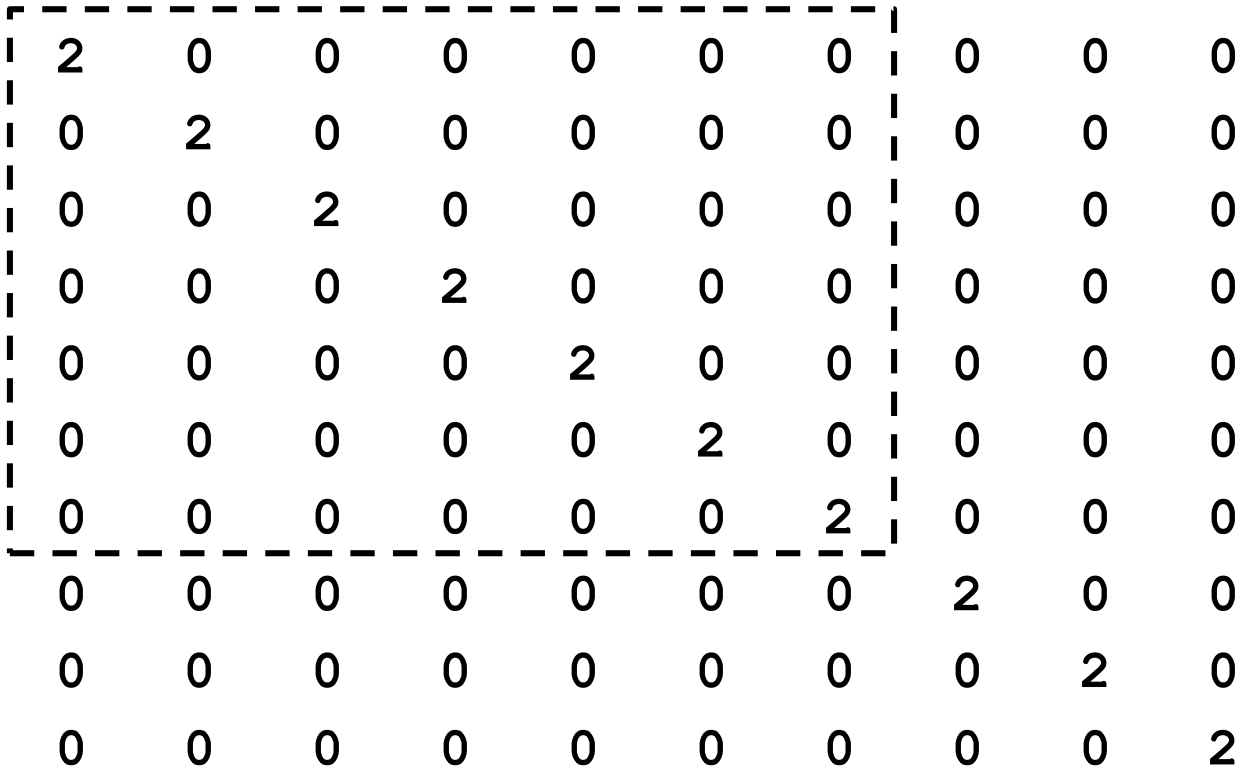
- O **programa incompleto** a seguir destina-se a:

  - **Ler** de um arquivo uma matriz quadrada **A** e escrevê-la no vídeo
  - **Pedir** e **receber** do operador dois inteiros **i** e **j**
  - **Calcular** e escrever no vídeo a matriz **MenorComplementar (A, i, j)**

- O referido arquivo terá uma **matriz quadrada 10 x 10**

- **Antes** de ler a matriz, o programa deve **pedir e receber** do operador a **dimensão** da matriz, que deve ser **≤ 10**

- Por exemplo, seja no próximo slide o conteúdo do arquivo



Completar, então, e executar, o programa a seguir

- Caso o operador digite **7** para a **dimensão** da matriz, deverão ser lidos apenas os **7 primeiros números** de cada linha

- O restante da linha deve ser **descartado** durante a leitura

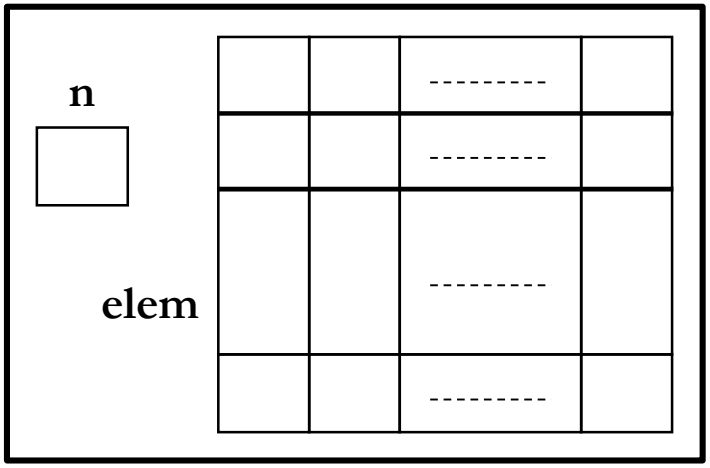
```
#include <stdio.h>
#include <stdlib.h>

typedef struct matquadrada matquadrada;
struct matquadrada {
   int n;  double elem[10][10];
};

void LerMatQuadrada (matquadrada *);
void EscreverMatQuadrada (matquadrada *);
matquadrada MenorComplementar (matquadrada *, int, int);
```

No programa, matrizes serão guardadas em estruturas

A passagem do argumento struct será por referência, pois será alterado



O tipo matquadrada

A passagem dos argumentos struct's será por referência para economia de memória

```
int main () {
   matquadrada A, B; int i, j;
   LerMatQuadrada (&A);
   printf ("\nMatriz A: \n\n");
   EscreverMatQuadrada (&A);
   printf ("\nMatriz Menor Complementar (i, j) de A:\n\n");
   printf ("Digite i e j: ");
   scanf ("%d%d", &i, &j);
   printf ("\n");
   B = MenorComplementar (&A, i, j);
   EscreverMatQuadrada (&B);
   printf("\n\n"); system ("pause");
   return 0;
}
```

O endereço de A é o argumento, pois a função vai alterar A

O endereço de A é o argumento, por economia, mas poderia ser o valor de A

O endereço de A é um dos argumentos, mas poderia se o valor de A

O valor retornado é uma estrutura matquadrada, que é atribuída a B

Na função main:

LerMatQuadrada (&A);

```
void LerMatQuadrada (matquadrada *Mat) {
   int i, j; char x;
   FILE *ff;
   ff = fopen ("<Nome do arquivo>", "r");
   printf ("Digite a dimensao n da matriz (1 <= n <= 10): ");
   scanf ("%d", &Mat->n);
   for (i = 0; i <= Mat->n-1; i++) {
       for (j = 0; j <= Mat->n-1; j++)
           fscanf (ff, "%f", &Mat->elem[i][j]);
       do
           fscanf (ff, "%c", &x);
       while (x != '\n');
   }
}
```

A matriz vai ser lida de um arquivo

A dimensão da matriz será digitada

`Mat->n` equivale a `(*Mat).n`

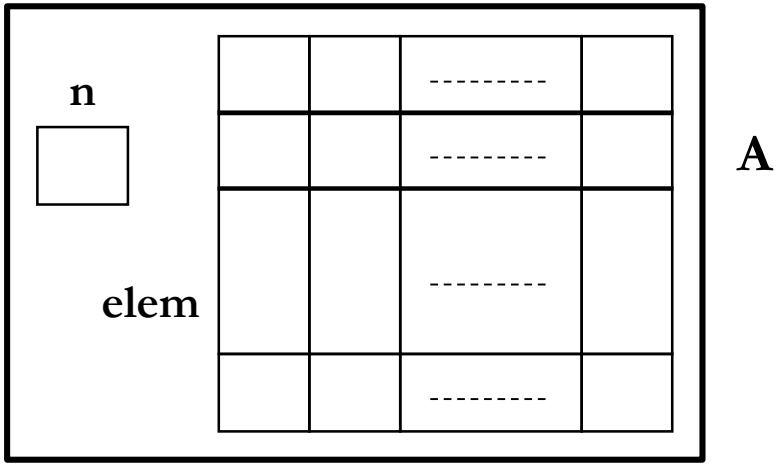
Ambos significam: campo n da estrutura cujo endereço está em Mat, ou, da estrutura apontada por Mat

```
void LerMatQuadrada (matquadrada *Mat) {
   int i, j; char c;
   FILE *ff;
   ff = fopen ("<Nome do arquivo>", "r");
   printf ("Digite a dimensao n da matriz (1 <= n <= 10): ");
   scanf ("%d", &Mat->n);
   for (i = 0; i <= Mat->n-1; i++) {
       for (j = 0; j <= Mat->n-1; j++)
           fscanf (ff, "%f", &Mat->elem[i][j]);
       do
           fscanf (ff, "%c", &x);
       while (x != '\n');
   }
}
```

Para descartar o resto de uma linha do arquivo de entrada

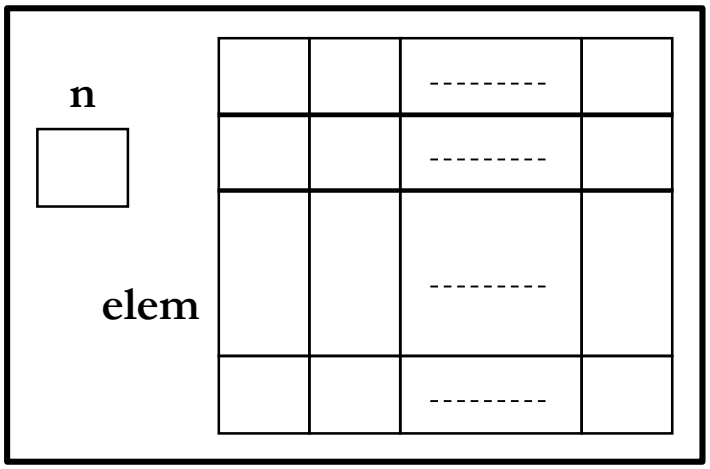
O local cujo endereço está em Mat é que vai receber os valores lidos

```
main:  LerMatQuadrada (&A);
```




```
void LerMatQuadrada (matquadrada *Mat) {
   - - - - - - - - -
   scanf ("%d", &Mat->n);
   for (i = 0; i <= Mat->n-1; i++) {
       for (j = 0; j <= Mat->n-1; j++)
           fscanf (ff, "%f", &Mat->elem[i][j]);
       - - - - - - - - -
   }
}
```

Chamada de LerMatQuadrada



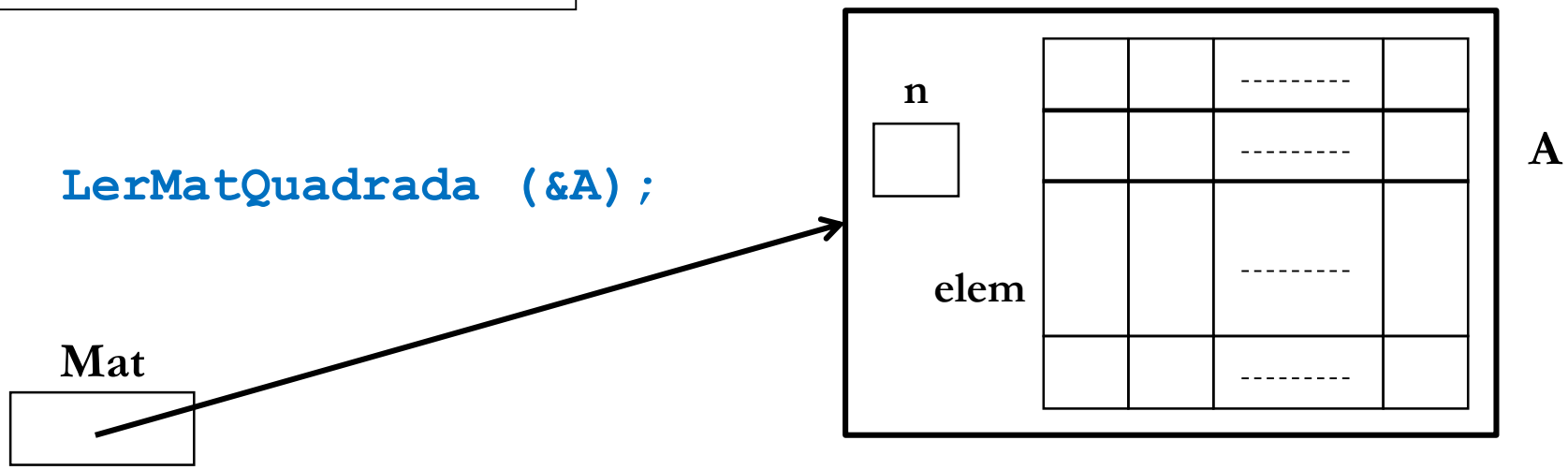

```
main:  LerMatQuadrada (&A);
```

Mat

```
void LerMatQuadrada (matquadrada *Mat) {
   - - - - - - - - -
   scanf ("%d", &Mat->n);
   for (i = 0; i <= Mat->n-1; i++) {
       for (j = 0; j <= Mat->n-1; j++)
           fscanf (ff, "%f", &Mat->elem[i][j]);
       - - - - - - - - -
   }
}
```

Passagem do argumento &A

```
main:  LerMatQuadrada (&A);
```




```
void LerMatQuadrada (matquadrada *Mat) {
   - - - - - - - - -
   scanf ("%d", &Mat->n);
   for (i = 0; i <= Mat->n-1; i++) {
       for (j = 0; j <= Mat->n-1; j++)
           fscanf (ff, "%f", &Mat->elem[i][j]);
       - - - - - - - - -
   }
}
```

```
void EscreverMatQuadrada (matquadrada *Mat) {
   - - - - -
}
```

Na função main:

EscreverMatQuadrada (&A);
EscreverMatQuadrada (&B);

```
matquadrada MenorComplementar (matquadrada *C,
                               int i, int j) {
   matquadrada D;
   - - - - -
   return D;
}
```

Na função main:

B = MenorComplementar (&A, i, j);

Em ambas as funções, a passagem da estrutura matquadrada poderia ser por valor, mas, para evitar transporte de estruturas da main para elas, essa passagem é por referência (endereço)

Exercício 10.5: Cálculo recursivo de determinantes por Laplace

- O cálculo do **determinante** de uma matriz quadrada **A** de dimensão **n** pode ter a seguinte formulação **recursiva**:

$$\text{Det}(A, n) = \begin{cases} A[0][0], \text{ se } n = 1 \\ A[0][0] * A[1][1] - A[1][0] * A[0][1], \text{ se } n = 2 \\ A[0][0] * A[1][1] * A[2][2] + A[0][1] * A[1][2] * A[2][0] + \\ A[0][2] * A[1][0] * A[2][1] - A[0][1] * A[1][0] * A[2][2] - \\ A[0][0] * A[1][2] * A[2][1] - A[0][2] * A[1][1] * A[2][0], \text{ se } n = 3 \text{ (Regra de Sarrus)} \\ \sum_{i=0}^{n-1} (-1)^i * A[0][i] * \text{Det}(\text{MenorComplementar}(A, 0, i), n - 1), \text{ se } n > 3 \\ \text{(Regra de Laplace)} \end{cases}$$

Livro 02

Escrever uma função recursiva **Det (A)** tal que:

- Seu parâmetro seja uma struct **matquadrada A**, conforme declarada no programa anterior, e que seu valor retornado seja o **determinante** de **A**, calculado pela formulação do slide anterior

- O argumento para a matriz **A** seja passado por referência, visando economia de memória

- A dimensão de **A** seja digitada pelo operador e recebida pela função, guardando-a no campo **n** de **A**

- Retorne um **valor atípico** (por exemplo, -999999999), se o valor digitado não for **positivo**

Fazer uma função **main** para:

- Ler a **dimensão** e os **elementos** de uma **matriz quadrada**
- Escrever no vídeo **esses elementos**
- Calcular e escrever o valor do **determinante** dessa matriz, chamando para isso a **função** descrita anteriormente

- A função **main** deve aproveitar a declaração de **matquadrada** e as funções **LerMatQuadrada**, **EscreverMatQuadrada** e **MenorComplementar**, elaboradas no exercício anterior

- A **matriz de entrada** para o cálculo do determinante deverá ser **lida de um arquivo**

- A digitação da **dimensão** pelo operador e sua leitura pelo programa, tem a finalidade de possibilitar sua **variação** entre diversas execuções

- Exemplo de conteúdo para o **arquivo** com a **matriz**:

```
 2   5  -4  -9  -3   5  -7   5   2   5
 0   3   5   0   4   4  -6   8  -3  -8
 7   2   1  -6   5   9   2  -7   4  -3
-1   6   0   8   3  -4   7  -2   9   5
 8  -3   9   0   5   9  -2   5   2   3
 7   3  -4  -3   1   9  -3   3  -5  -2
 9   3  -2   1   0   4   2  -5   3   1
 3  -1   0   7   2  -3   7   8  -2   7
-5   9   2  -4  -3   0   9   2   5   2
 5  -4   1   8   6  -2   9  -9  -1   9
```

- Caso o operador digite uma **dimensão** menor que 10, **7**, por exemplo, deverão ser lidos apenas os **7 primeiros números** de cada linha

Exercício 10.6: Instrumentação do programa dos determinantes

- Fazer alterações e inclusões no programa do exercício anterior para montar uma **tabela** com:
  - o número de (**somas + subtrações**) de números reais
  - e com o número de **multiplicações** de números reais

  realizadas pelo programa, em função da **dimensão** da matriz de entrada

- Então, em vez da **digitação** da dimensão, esta deverá **variar de 1 a 10**, numa mesma execução

- Só contabilizar uma operação se os **dois** operandos forem **reais**

A seguir, um exemplo de saída para o programa instrumentado

```
Calculo de determinante com instrumentacao

Matriz com dimensao 1:

|    2|

Valor do determinante:                    2


Matriz com dimensao 2:

|    2    5|
|    0    3|

Valor do determinante:                    6


Matriz com dimensao 3:

|    2    5   -4|
|    0    3    5|
|    7    2    1|

Valor do determinante:                  245


Matriz com dimensao 4:

|    2    5   -4   -9|
|    0    3    5    0|
|    7    2    1   -6|
|   -1    6    0    8|

Valor do determinante:                 3331
```

Matriz com dimensao 5:

```
|    2    5   -4   -9   -3|
|    0    3    5    0    4|
|    7    2    1   -6    5|
|   -1    6    0    8    3|
|    8   -3    9    0    5|
```

Valor do determinante: **-24173**

Matriz com dimensao 6:

```
|    2    5   -4   -9   -3    5|
|    0    3    5    0    4    4|
|    7    2    1   -6    5    9|
|   -1    6    0    8    3   -4|
|    8   -3    9    0    5    9|
|    7    3   -4   -3    1    9|
```

Valor do determinante: **-125629**

Matriz com dimensao 7:

```
|    2    5   -4   -9   -3    5   -7|
|    0    3    5    0    4    4   -6|
|    7    2    1   -6    5    9    2|
|   -1    6    0    8    3   -4    7|
|    8   -3    9    0    5    9   -2|
|    7    3   -4   -3    1    9   -3|
|    9    3   -2    1    0    4    2|
```

Valor do determinante: **688135**

Matriz com dimensao 8:

```
|    2    5   -4   -9   -3    5   -7    5|
|    0    3    5    0    4    4   -6    8|
|    7    2    1   -6    5    9    2   -7|
|   -1    6    0    8    3   -4    7   -2|
|    8   -3    9    0    5    9   -2    5|
|    7    3   -4   -3    1    9   -3    3|
|    9    3   -2    1    0    4    2   -5|
|    3   -1    0    7    2   -3    7    8|
```

Valor do determinante: 13284312

Matriz com dimensao 9:

```
|    2    5   -4   -9   -3    5   -7    5    2|
|    0    3    5    0    4    4   -6    8   -3|
|    7    2    1   -6    5    9    2   -7    4|
|   -1    6    0    8    3   -4    7   -2    9|
|    8   -3    9    0    5    9   -2    5    2|
|    7    3   -4   -3    1    9   -3    3   -5|
|    9    3   -2    1    0    4    2   -5    3|
|    3   -1    0    7    2   -3    7    8   -2|
|   -5    9    2   -4   -3    0    9    2    5|
```

Valor do determinante: -668899744

```
Matriz com dimensao 10:

|    2    5   -4   -9   -3    5   -7    5    2    5|
|    0    3    5    0    4    4   -6    8   -3   -8|
|    7    2    1   -6    5    9    2   -7    4   -3|
|   -1    6    0    8    3   -4    7   -2    9    5|
|    8   -3    9    0    5    9   -2    5    2    3|
|    7    3   -4   -3    1    9   -3    3   -5   -2|
|    9    3   -2    1    0    4    2   -5    3    1|
|    3   -1    0    7    2   -3    7    8   -2    7|
|   -5    9    2   -4   -3    0    9    2    5    2|
|    5   -4    1    8    6   -2    9   -9   -1    9|

Valor do determinante:          -9804411118

Numeros de somas+subtracoes e multiplicacoes:

Dimensao:  1            0 somas+subtracoes        0 multiplicacoes
Dimensao:  2            1 somas+subtracoes        2 multiplicacoes
Dimensao:  3            5 somas+subtracoes       12 multiplicacoes
Dimensao:  4           24 somas+subtracoes       52 multiplicacoes
Dimensao:  5          125 somas+subtracoes      265 multiplicacoes
Dimensao:  6          756 somas+subtracoes     1596 multiplicacoes
Dimensao:  7         5299 somas+subtracoes    11179 multiplicacoes
Dimensao:  8        42400 somas+subtracoes    89440 multiplicacoes
Dimensao:  9       381609 somas+subtracoes   804969 multiplicacoes
Dimensao: 10      3816100 somas+subtracoes  8049700 multiplicacoes
```

AULAS PRÁTICAS DE INTRODUÇÃO À PROGRAMAÇÃO

Capítulo XI

Encadeamento de Estruturas por Ponteiros

Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

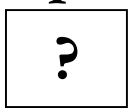
int main () {
   st *p;
```

Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
   st *p;
```

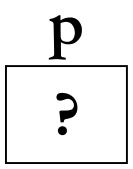


Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
  st *p;
  p = (st *) malloc (sizeof(st));
```

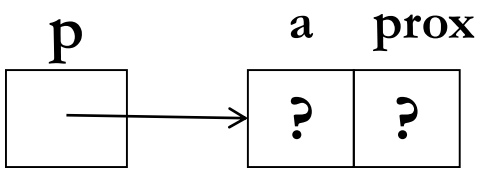


Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
  st *p;
  p = (st *) malloc (sizeof(st));
```

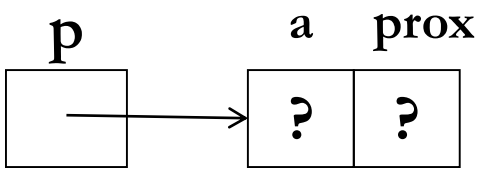


Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
  st *p;
  p = (st *) malloc (sizeof(st));
  p->a = 2;
```

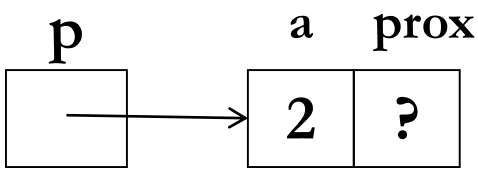


Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
   st *p;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
```

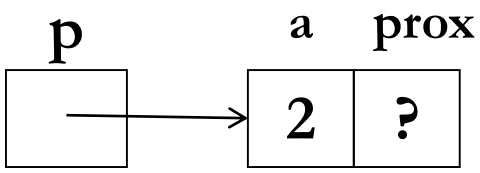


Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
   st *p;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
   printf ("%4d", p->a);
```

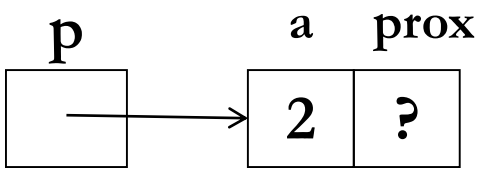


Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
   st *p;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
   printf ("%4d", p->a);
```



Vídeo

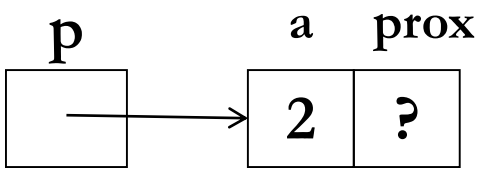
2

Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
   st *p;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
   printf ("%4d", p->a);
   printf ("\n\n"); system ("pause"); return 0;
}
```



Vídeo

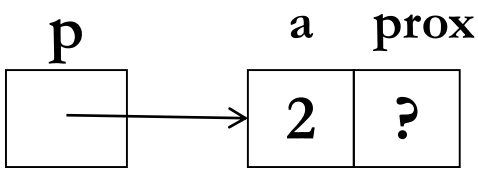
```
2
```

Programa 11.1: Alocação, preenchimento e escrita de uma nova estrutura

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

typedef struct st st;
struct st {int a; st *prox;};

int main () {
   st *p;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
   printf ("%4d", p->a);
   printf ("\n\n"); system ("pause"); return 0;
}
```



Vídeo

2

Copiar, salvar e executar

Programa 11.2: Alocação, preenchimento e escrita de 3 novas estruturas

```
typedef struct st st;
struct st {int a; st *prox;};
typedef st *pontst;

int main () {
   pontst p, q;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
   p->prox = (st *) malloc (sizeof(st));
   p->prox->a = 3;
   p->prox->prox = (st *) malloc (sizeof(st));
   p->prox->prox->a = 5;
   p->prox->prox->prox = NULL;
   for (q = p; q != NULL; q = q->prox)
       printf ("%4d", q->a);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Copiar, salvar e executar

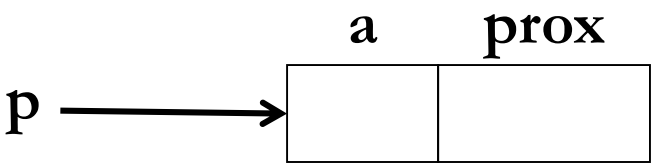
- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

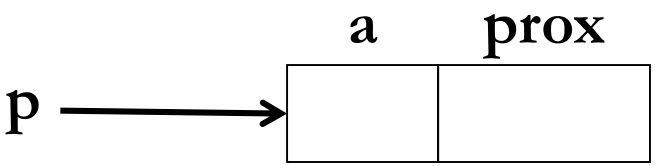


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

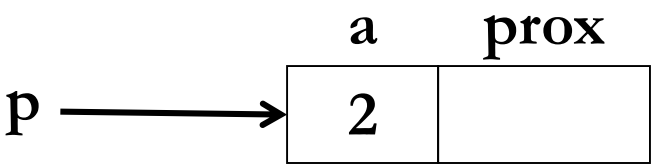


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

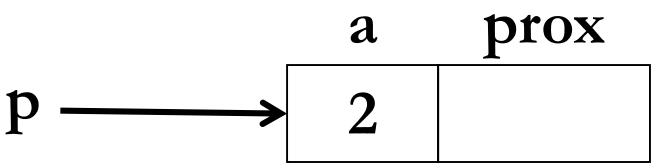


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

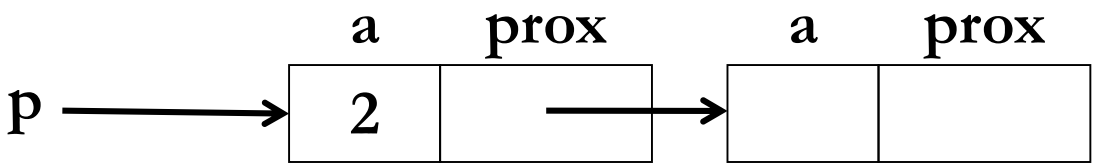


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

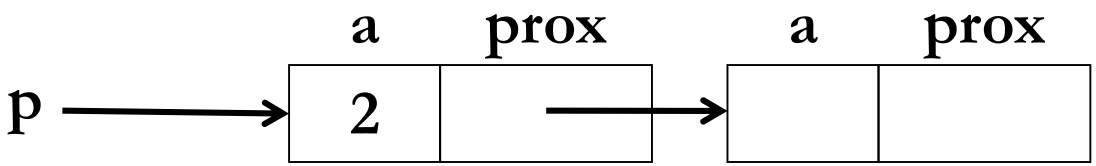


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

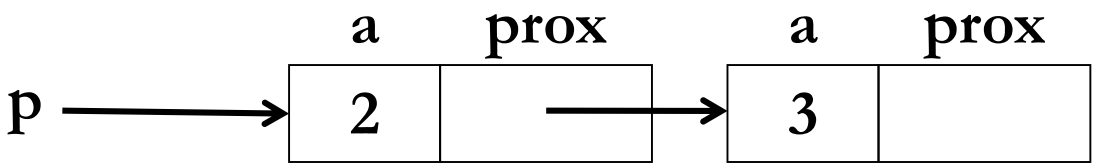


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

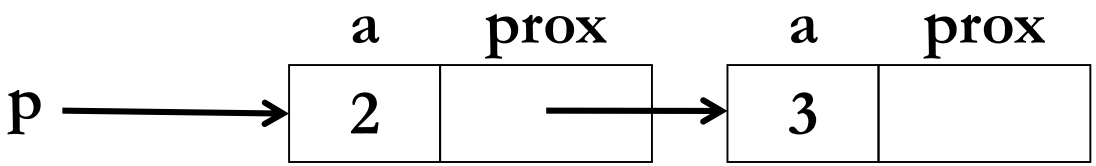


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

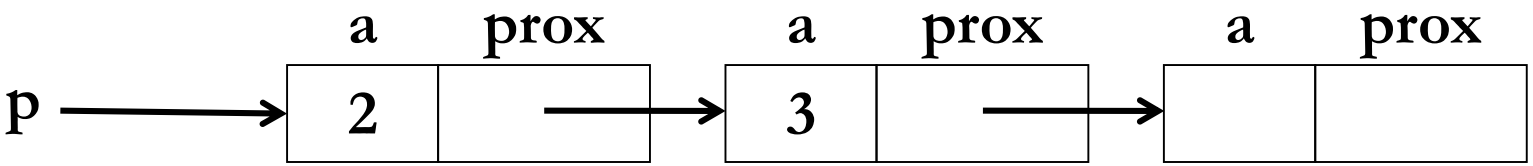


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

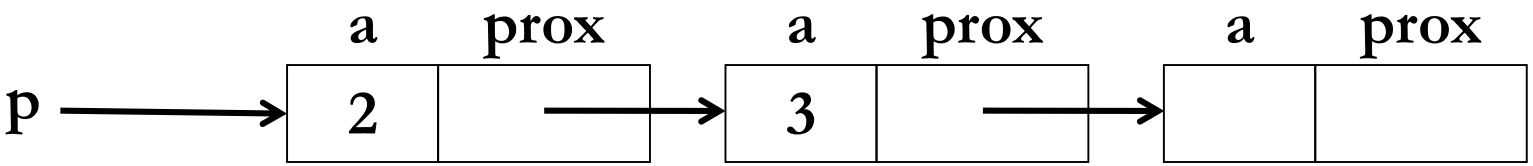


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

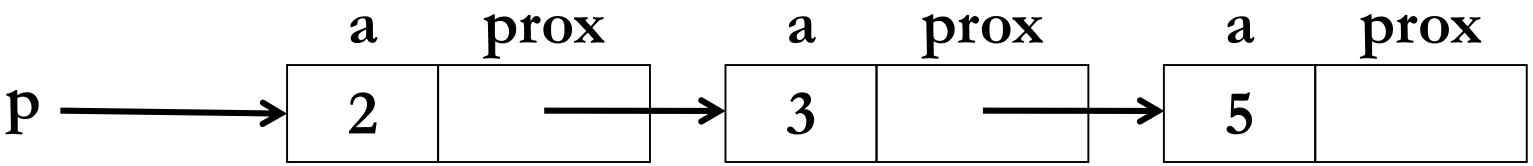


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

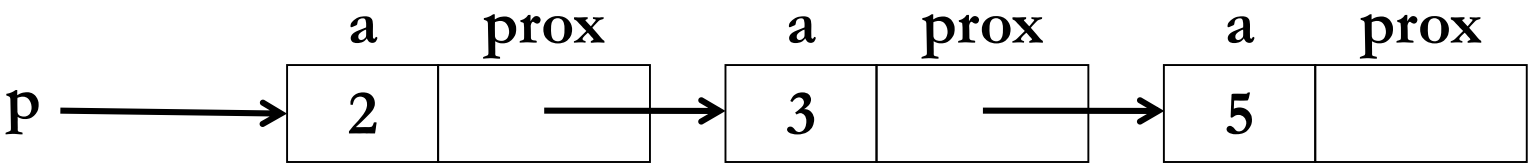


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

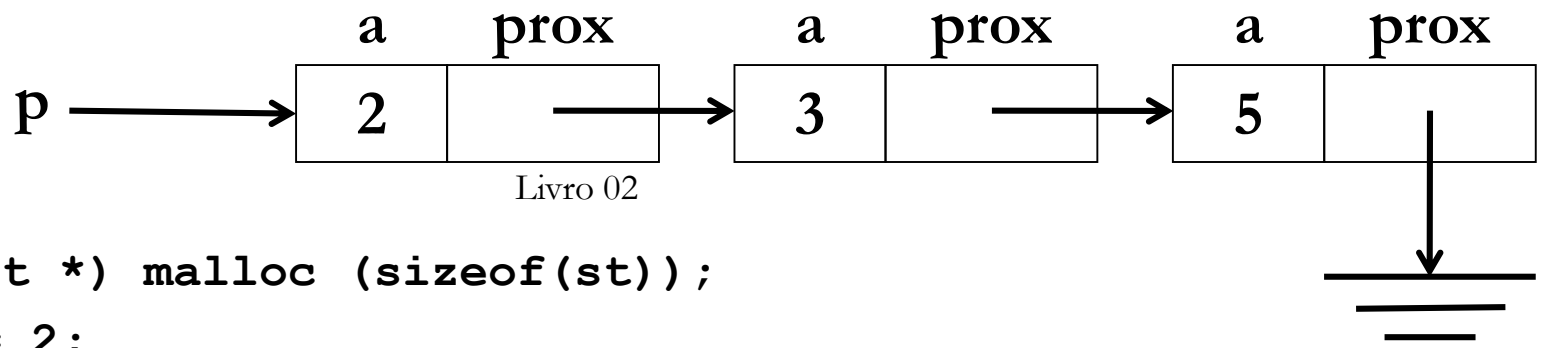


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```



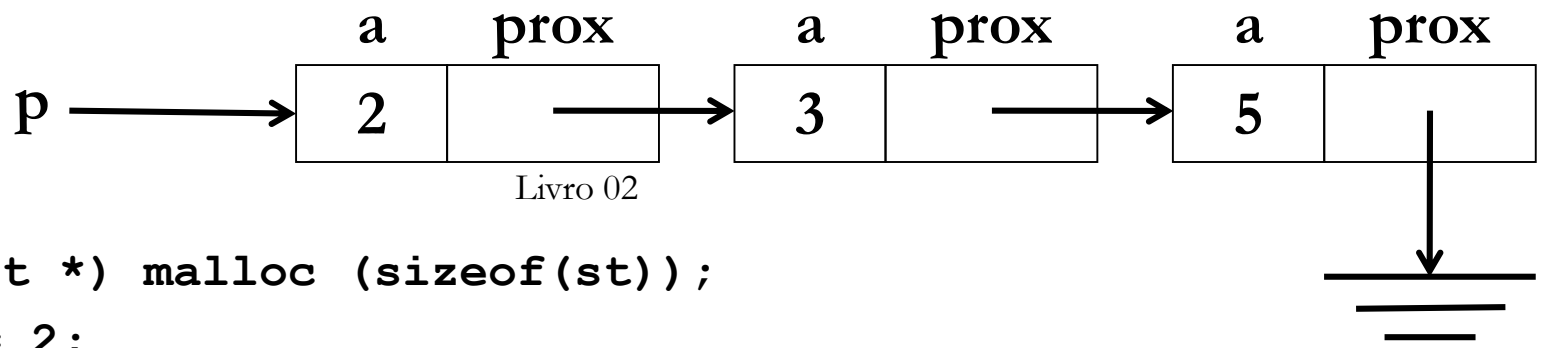
Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

Formou-se um encadeamento de estruturas

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

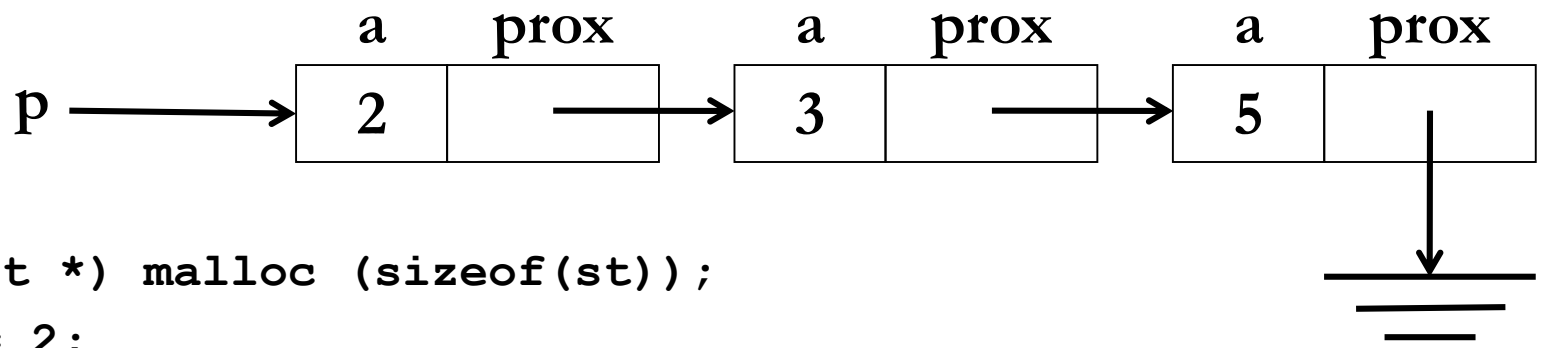


Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

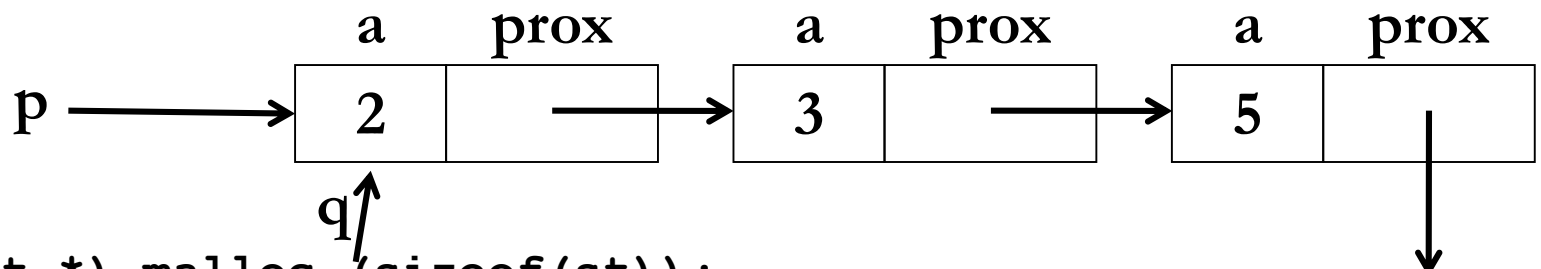



```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

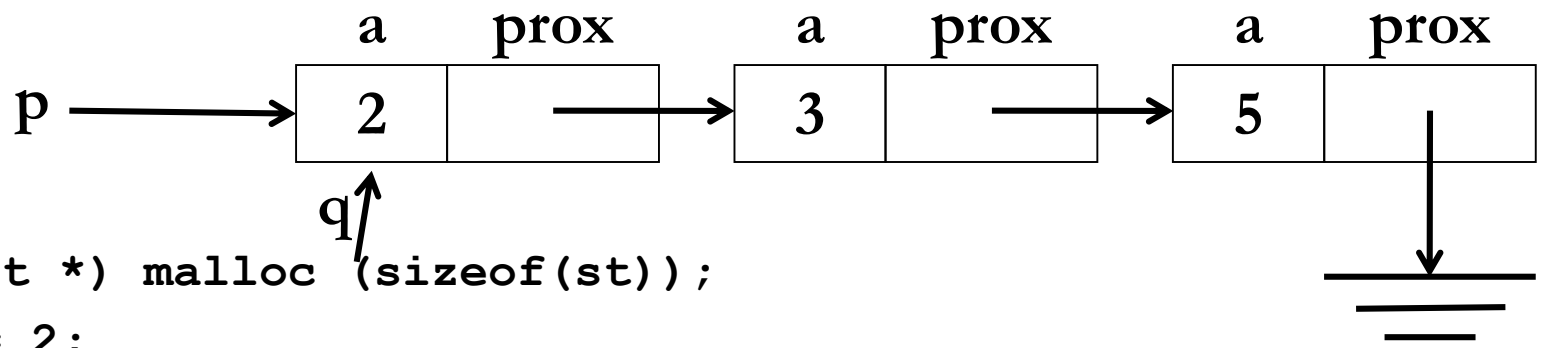



```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

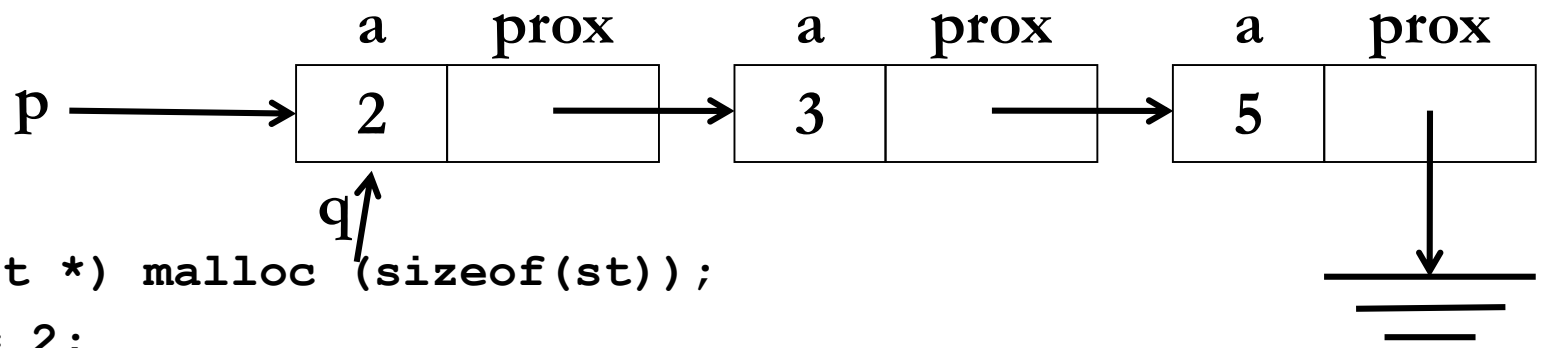



```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

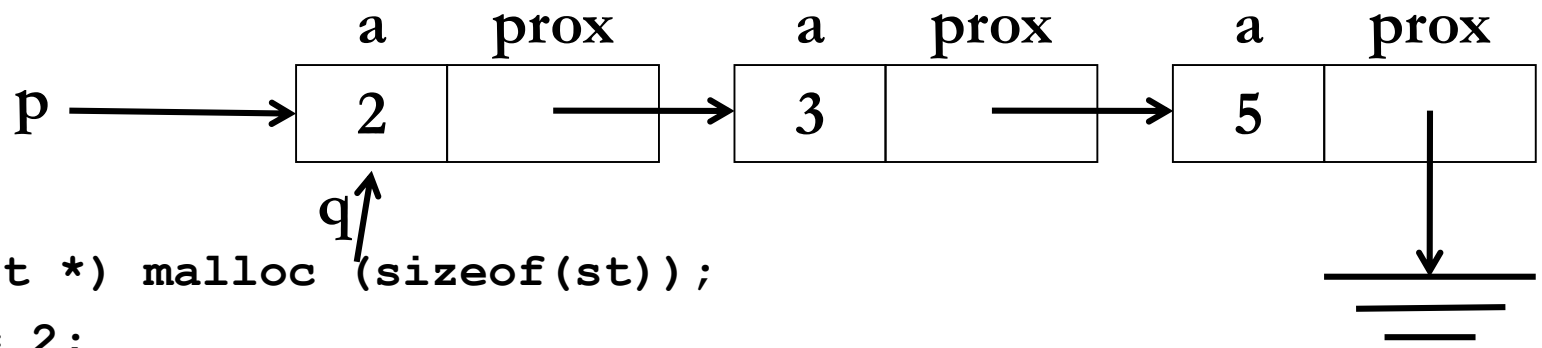
Vídeo

2

PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

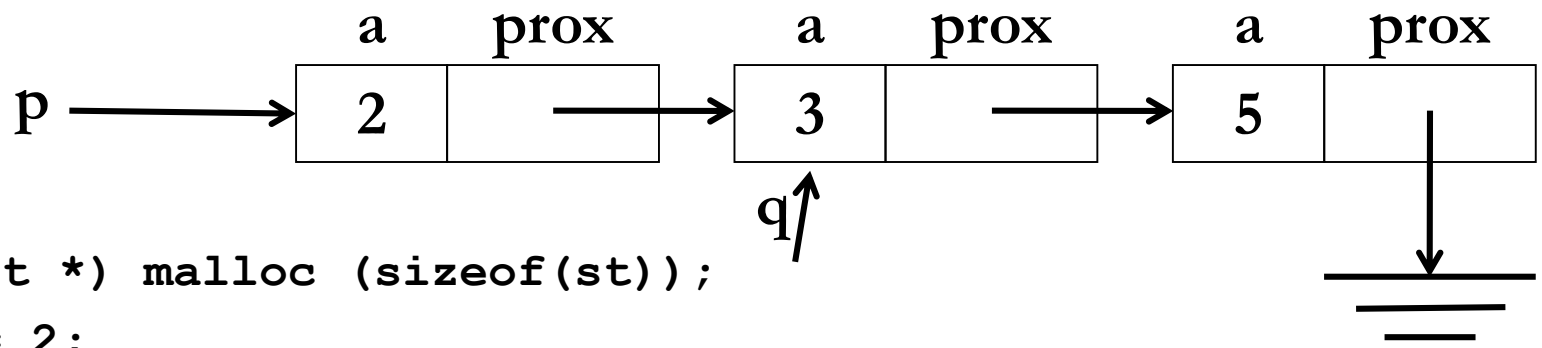
Vídeo

2

PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

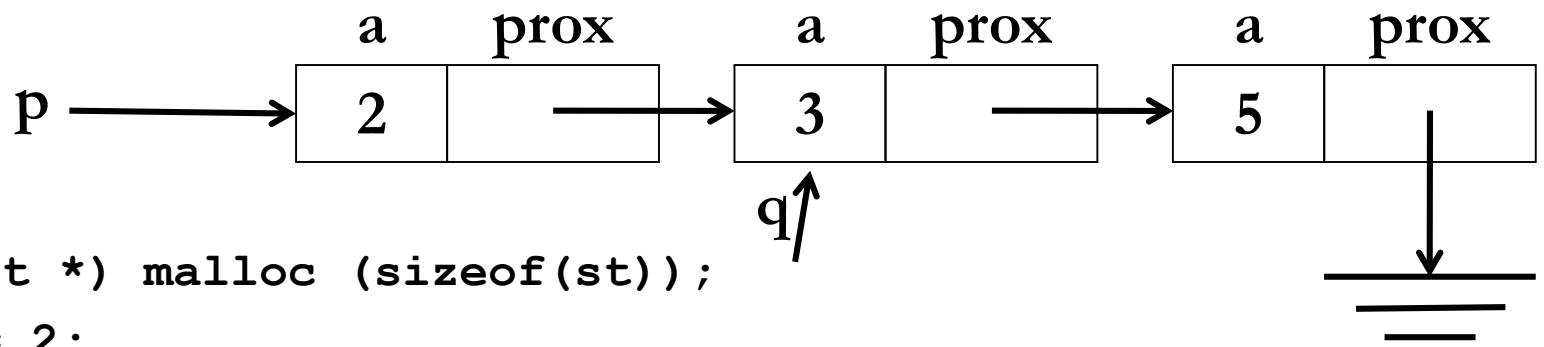
Vídeo

2

PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

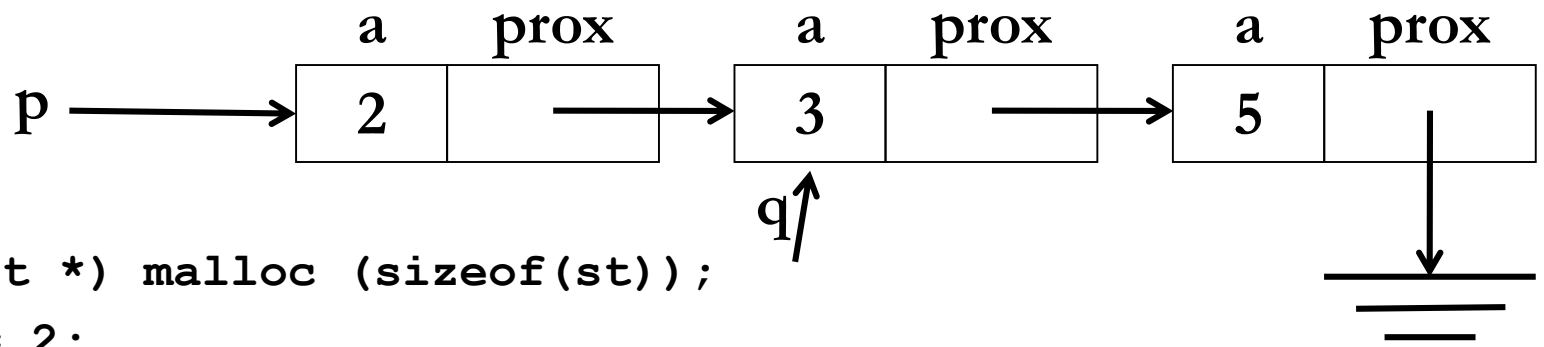
Vídeo

2

PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

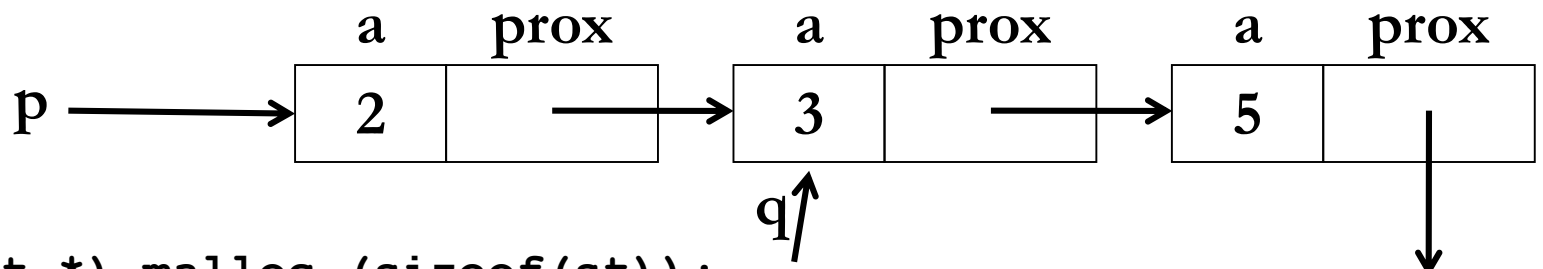
Vídeo



PPT 01

- Seja a execução dos comandos:

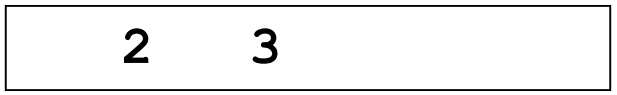
```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

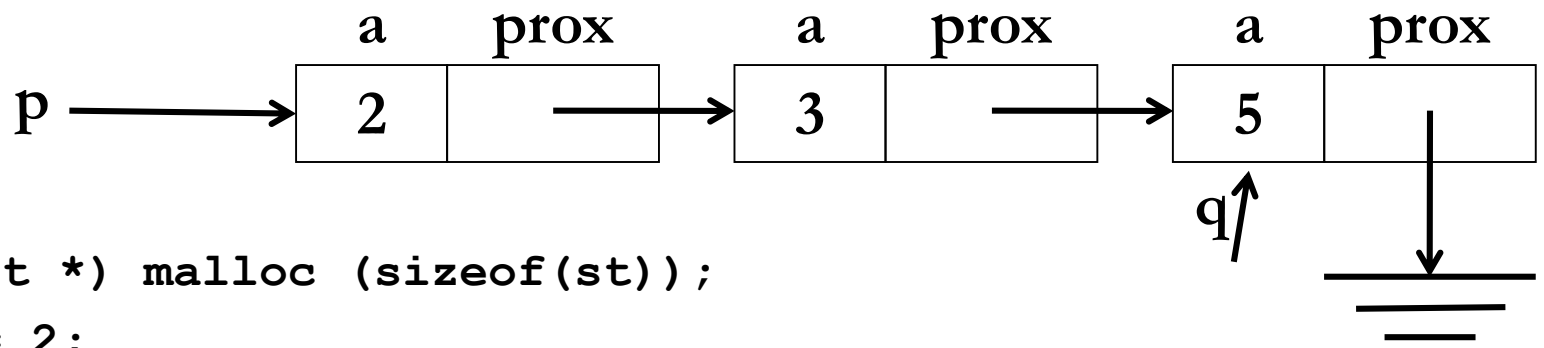
Vídeo



PPT 01

- Seja a execução dos comandos:

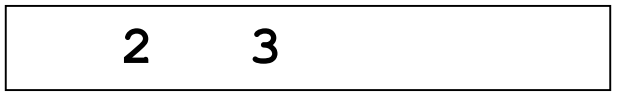
```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

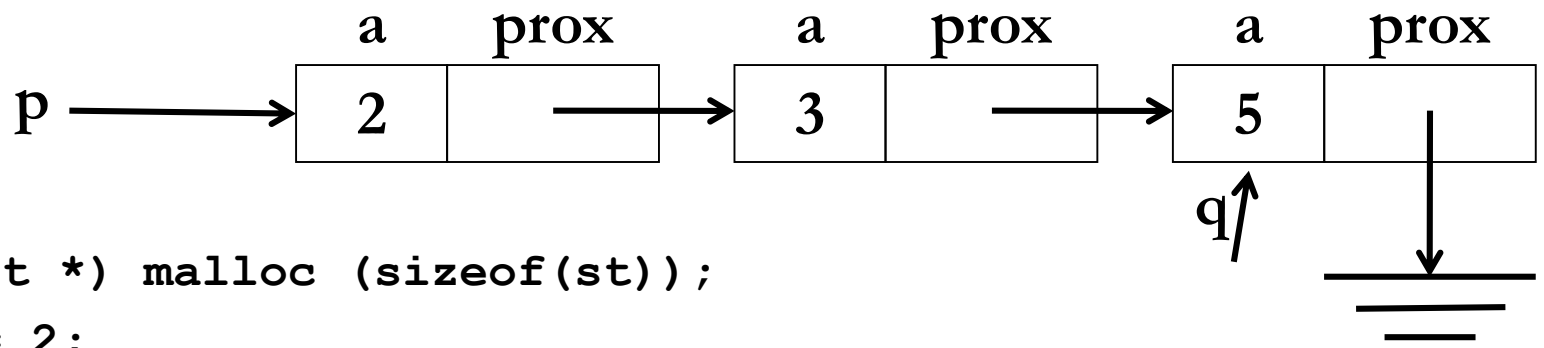
Vídeo



PPT 01

- Seja a execução dos comandos:

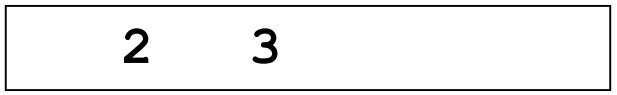
```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

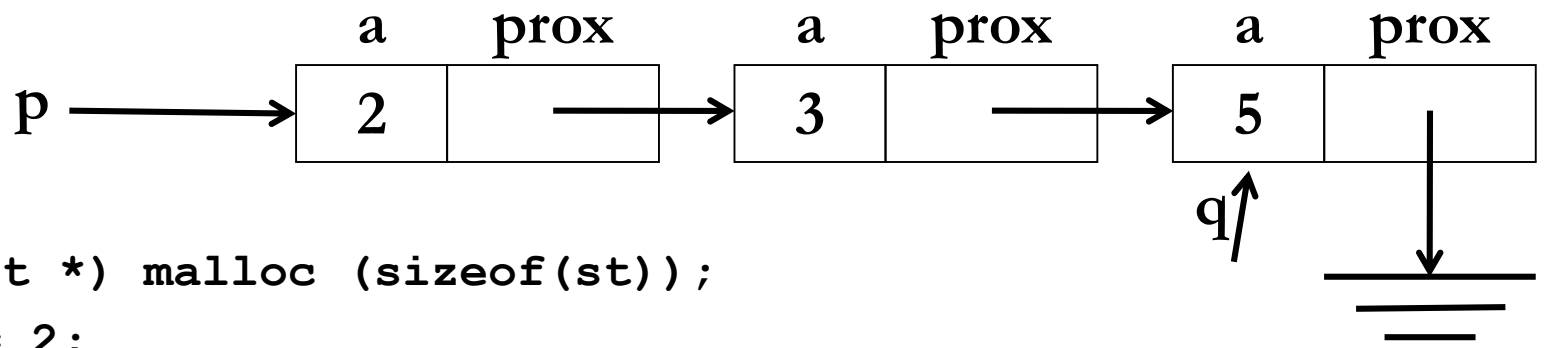
Vídeo



PPT 01

- Seja a execução dos comandos:

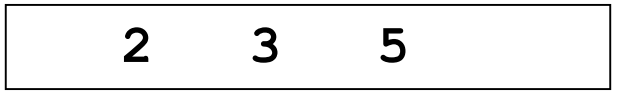
```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

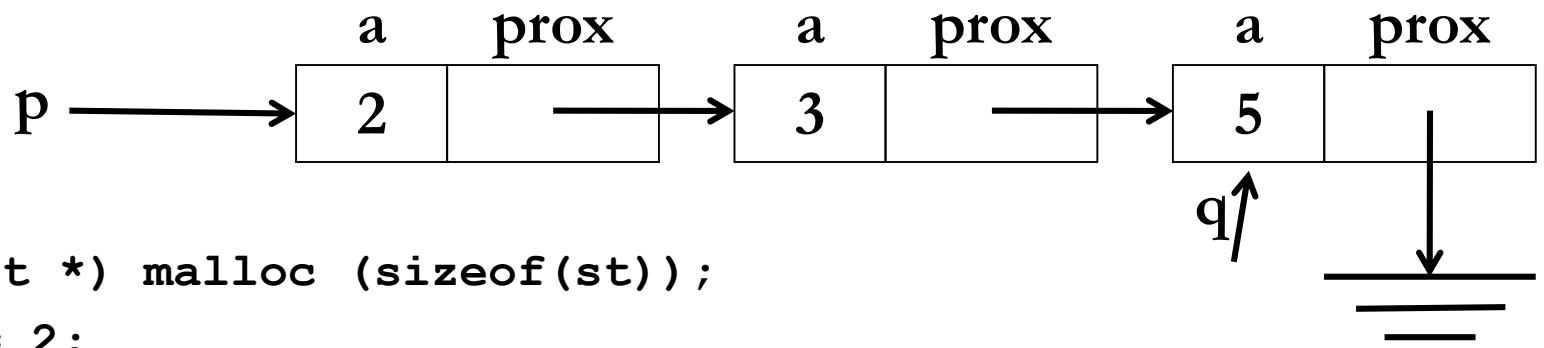
Vídeo



PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

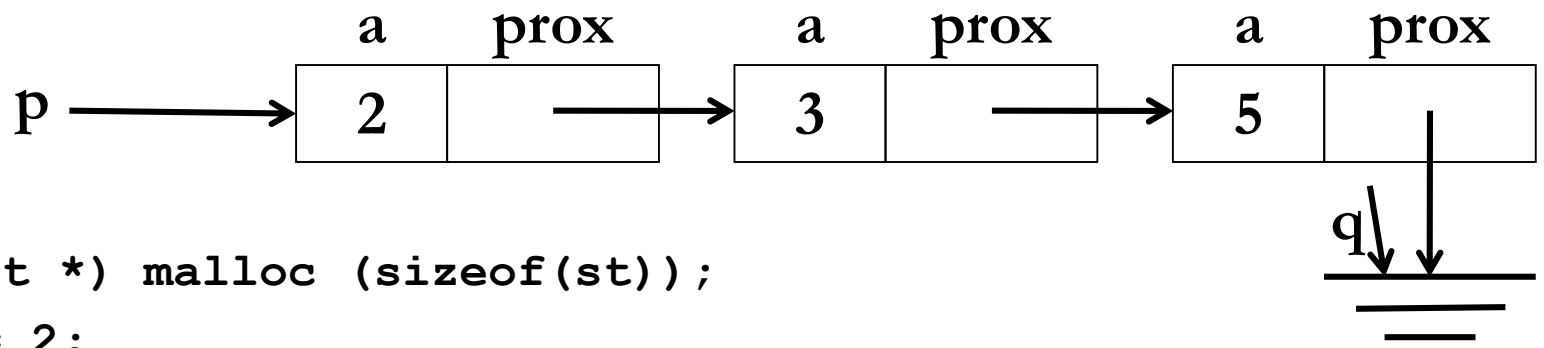
Vídeo



PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```



Livro 02

```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Vídeo



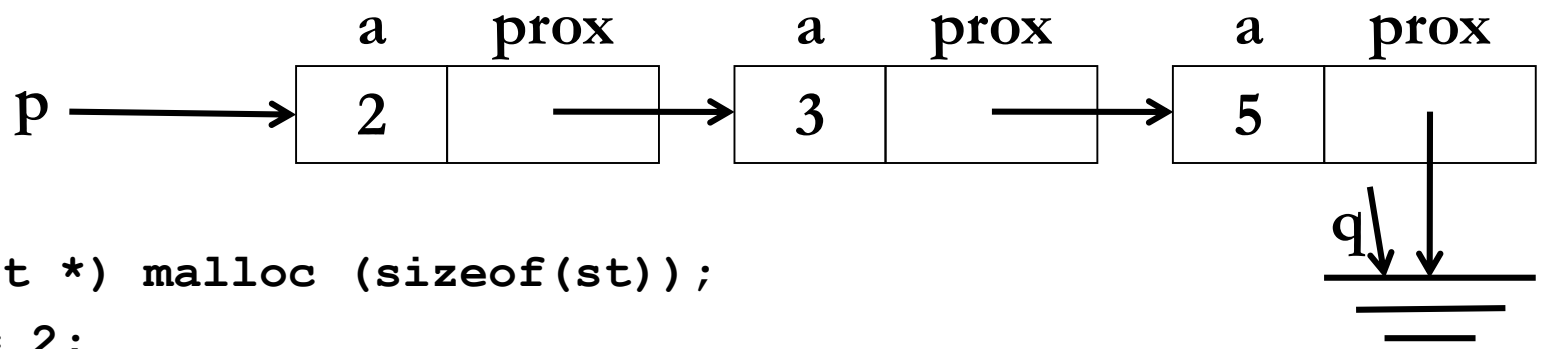
PPT 01

- Seja a execução dos comandos:

```
typedef struct st st;
struct st {int a; st *prox;};
st *p, *q;
```

O ponteiro q percorreu o encadeamento com o comando

```
q = q->prox
```




```
p = (st *) malloc (sizeof(st));
p->a = 2;
p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Livro 02

q == NULL:

Fim!!!

Vídeo

2 3 5

PPT 01

Programa 11.3: Deixando vazia a primeira estrutura (chamando-a de **estrutura-líder**)

```
typedef struct st st;
struct st {int a; st *prox;};
typedef st *pontst;

int main () {
   pontst p, q;
   p = (st *) malloc (sizeof(st));
   p->a = 2;
   p->prox = (st *) malloc (sizeof(st));
   p->prox->a = 3;
   p->prox->prox = (st *) malloc (sizeof(st));
   p->prox->prox->a = 5;
   p->prox->prox->prox = NULL;
   for (q = p; q != NULL; q = q->prox)
       printf ("%4d", q->a);
   printf ("\n\n"); system ("pause"); return 0;
}
```

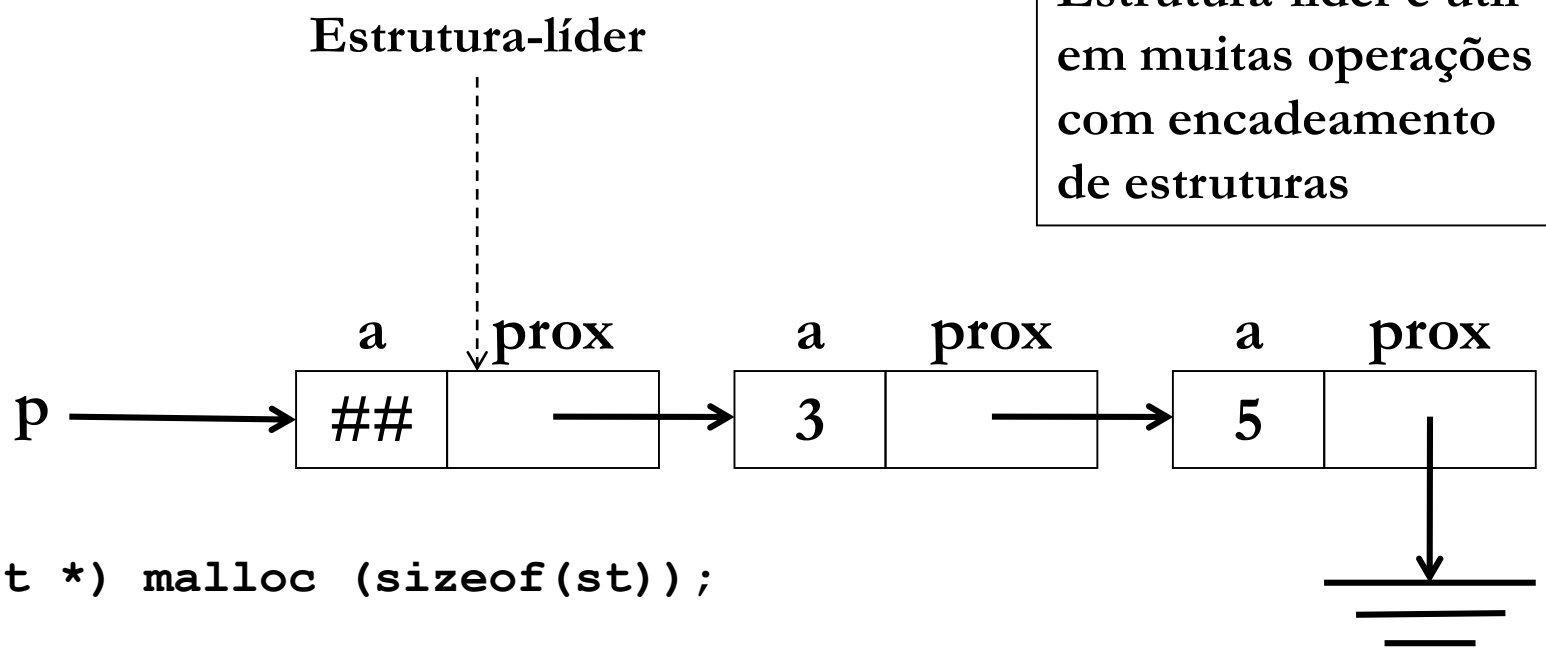
Programa 11.3: Deixando vazia a primeira estrutura (chamando-a de **estrutura-líder**)

```
typedef struct st st;
struct st {int a; st *prox;};
typedef st *pontst;

int main () {
   pontst p, q;
   p = (st *) malloc (sizeof(st));
              <----------------------------------------------------
   p->prox = (st *) malloc (sizeof(st));
   p->prox->a = 3;
   p->prox->prox = (st *) malloc (sizeof(st));
   p->prox->prox->a = 5;
   p->prox->prox->prox = NULL;
   for (q = p->prox; q != NULL; q = q->prox)
       printf ("%4d", q->a);
   printf ("\n\n"); system ("pause"); return 0;
}
```

Copiar, salvar e executar

Estrutura-líder é útil em muitas operações com encadeamento de estruturas




```
p = (st *) malloc (sizeof(st));

p->prox = (st *) malloc (sizeof(st));
p->prox->a = 3;
p->prox->prox = (st *) malloc (sizeof(st));
p->prox->prox->a = 5;
p->prox->prox->prox = NULL;
for (q = p->prox; q != NULL; q = q->prox)
  printf ("%4d", q->a);
```

Vídeo

```
   3   5
```

Programa 11.4: Lendo o nº de estruturas e os valores a serem nelas guardados (mantendo a estrutura-líder)

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

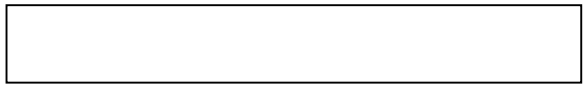
/* Declaracoes dos tipos */

typedef struct st st;
struct st {int a; st *prox;};
typedef st *pontst;

/* main: cabecalho e declaracoes locais */

int main () {
   int i, n;
   pontst p, q;
```

Buffer do teclado



i **n**

p

```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));
```

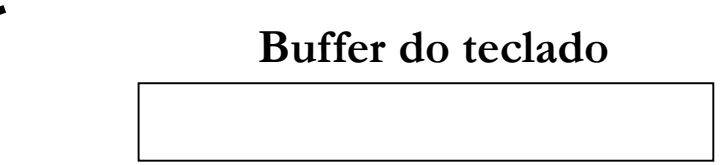






```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));
```

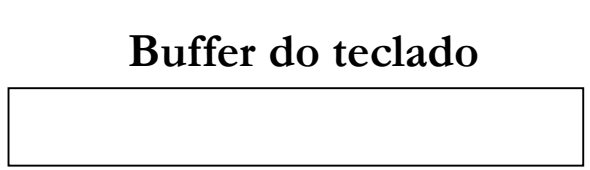




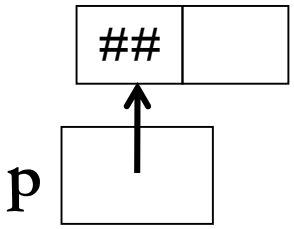


```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));
```





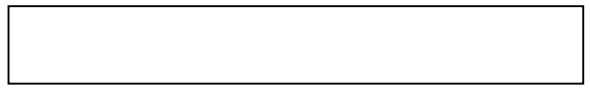


```
/* Alocacao da estrutura-lider  */

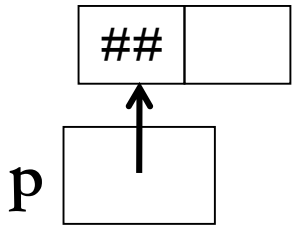
   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);
```







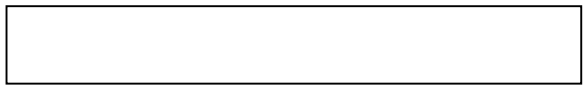
```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

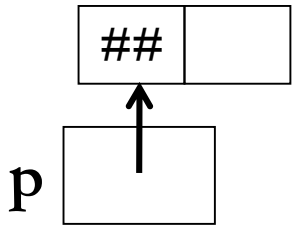
/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

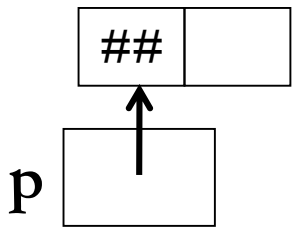
/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);
```

Buffer do teclado

3

i		n	



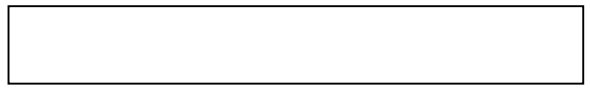
```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

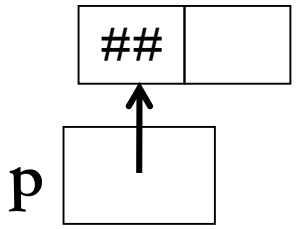
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);
```

Buffer do teclado



i

n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

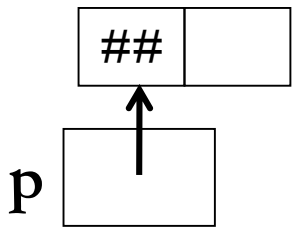
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i

n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

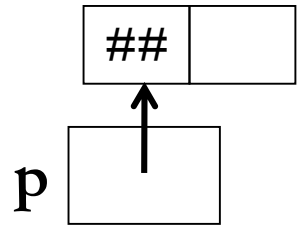
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i

n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

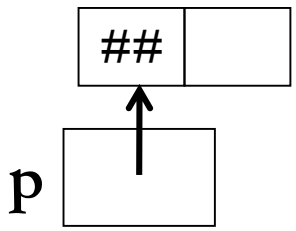
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i

n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

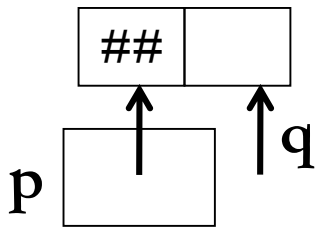
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

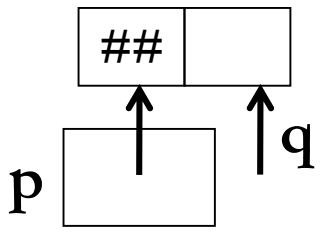
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

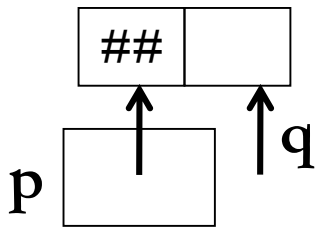
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i 1 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

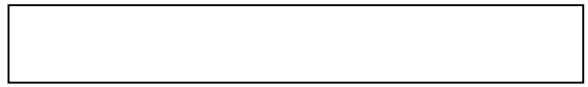
/* Leitura do numero de estruturas */

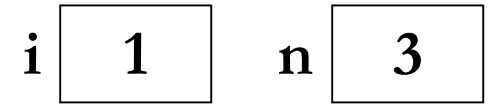
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

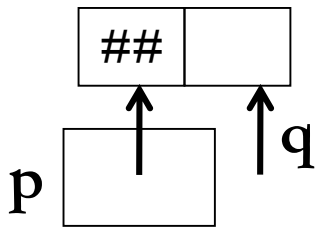
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

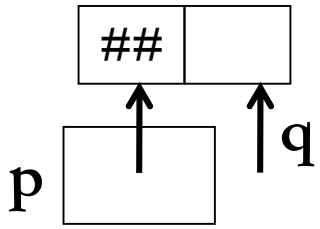
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
    - - - - > q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i 1 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

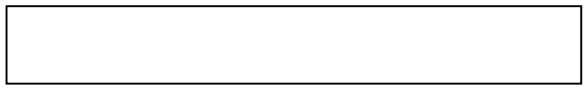
/* Leitura do numero de estruturas */

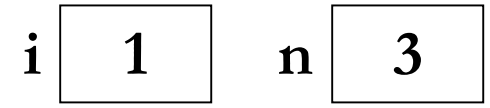
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

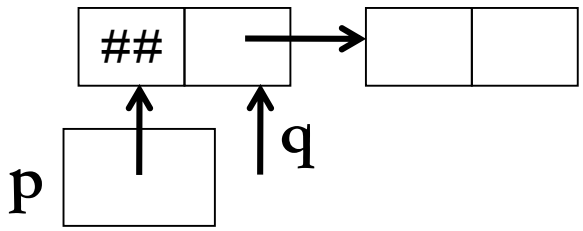
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
    - - - - > q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

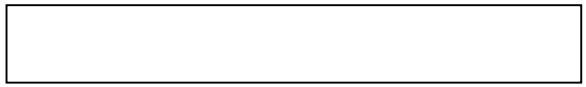
/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

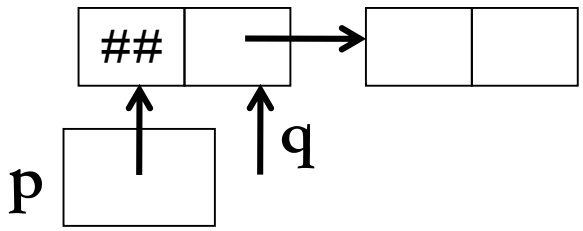
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
        - - - - > q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado



i 1 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

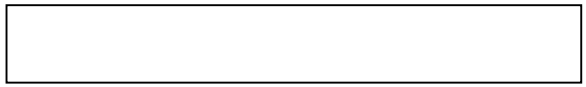
/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

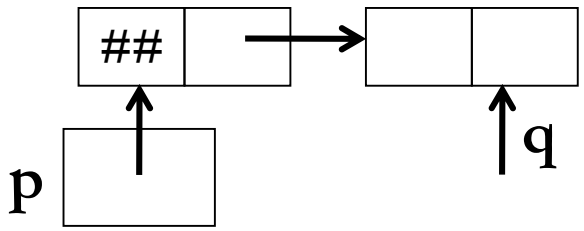
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
        - - - - > q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado



i 1 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

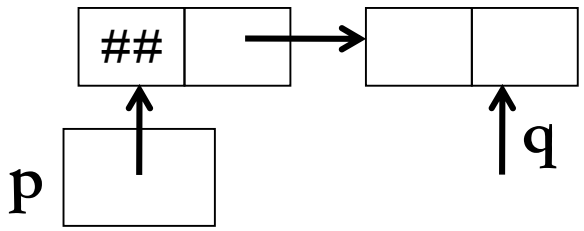
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
    - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

i 1 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

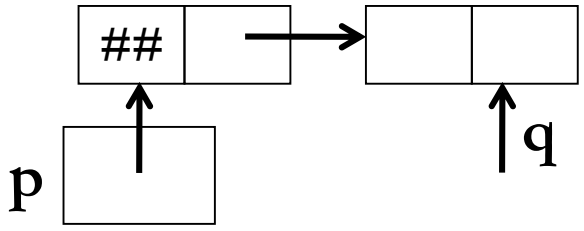
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
    - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

2	5	8

i | 1 | n | 3 |



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

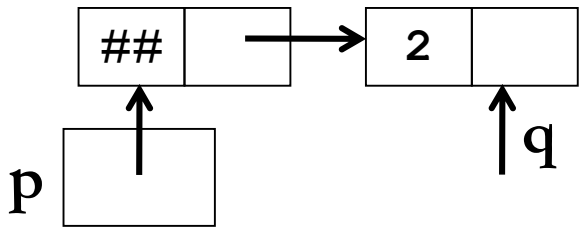
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
     - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i	1

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

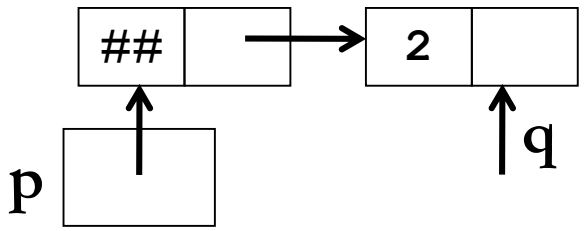
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i 1 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

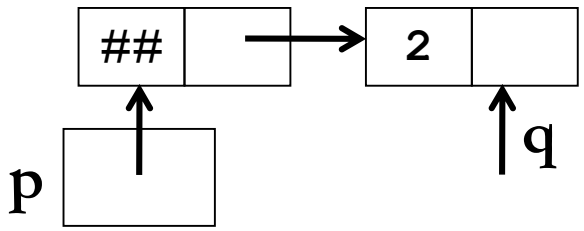
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i 2 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

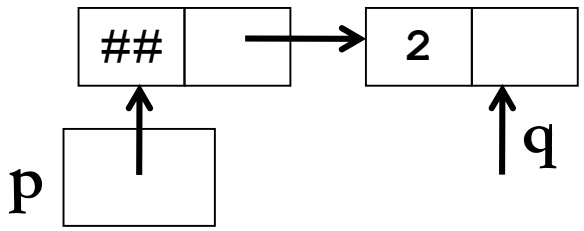
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i	2

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

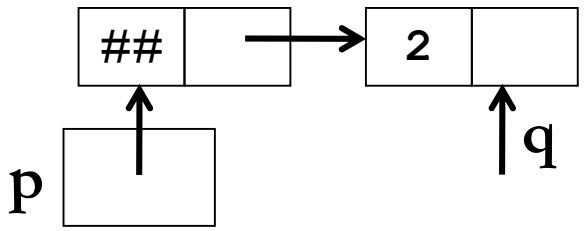
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
    - - - - > q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i	2

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

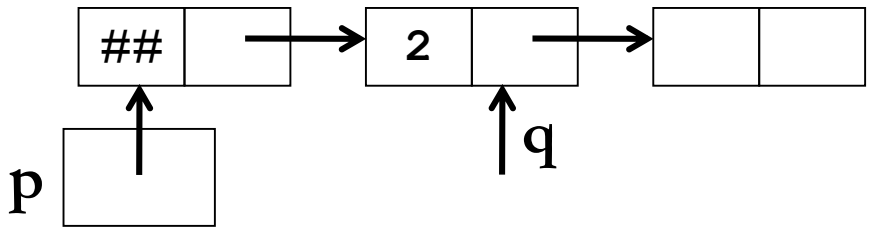
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
    - - - - > q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i 2 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

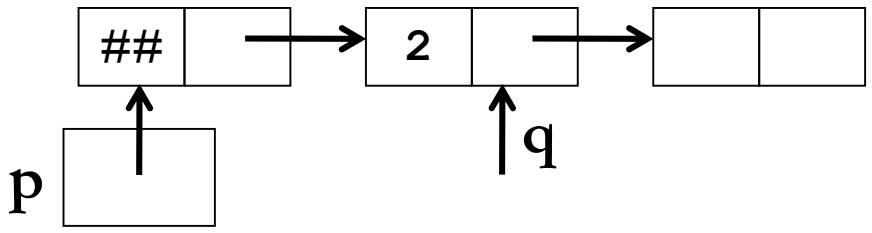
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
        - - - - > q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i | 2 | n | 3 |



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

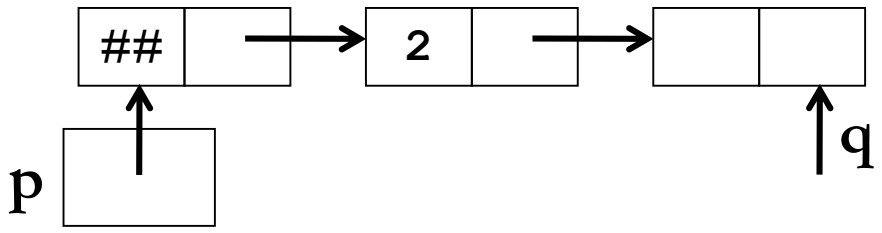
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
        - - - - > q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i	2

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

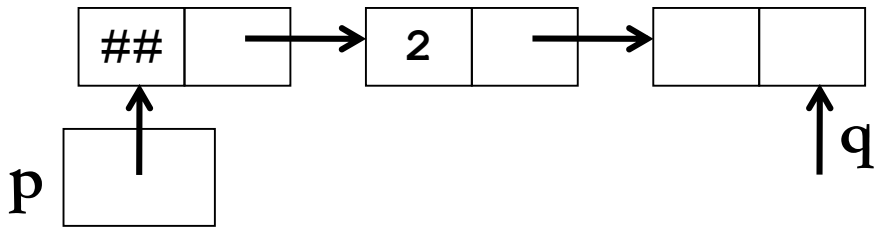
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
    - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

5	8

i 2 n 3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

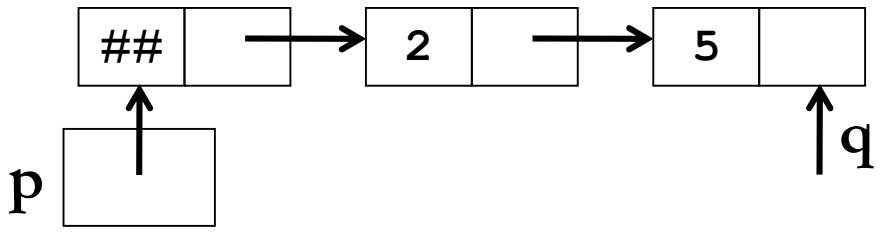
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
    - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i | 2 | n | 3 |



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

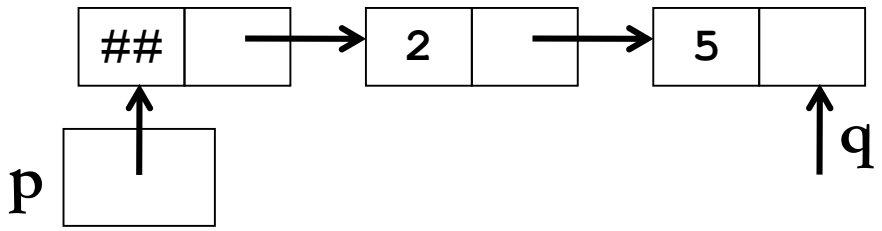
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	2

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

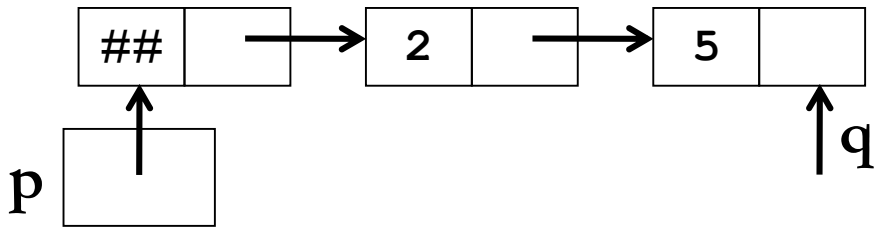
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

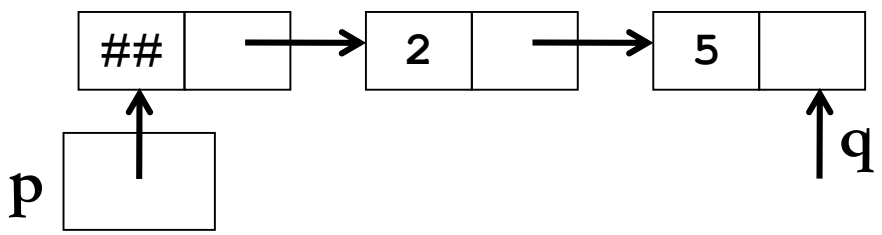
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

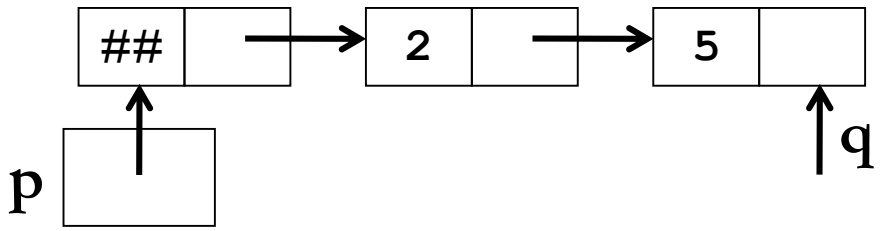
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
    - - - - > q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

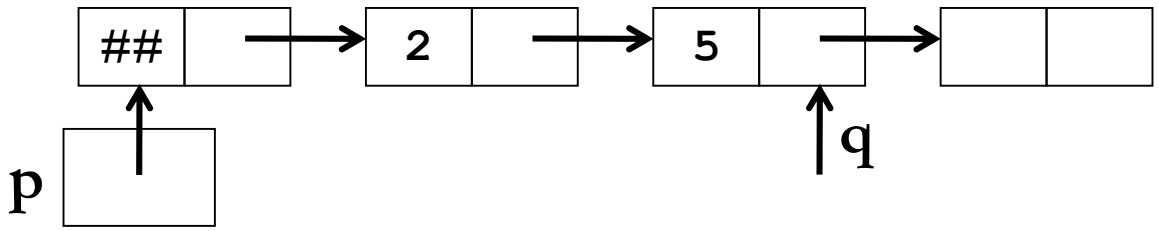
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
    - - - - > q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

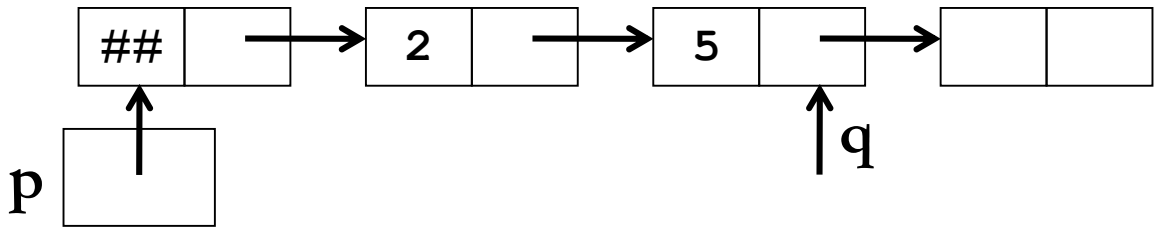
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
    - - - - >   q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

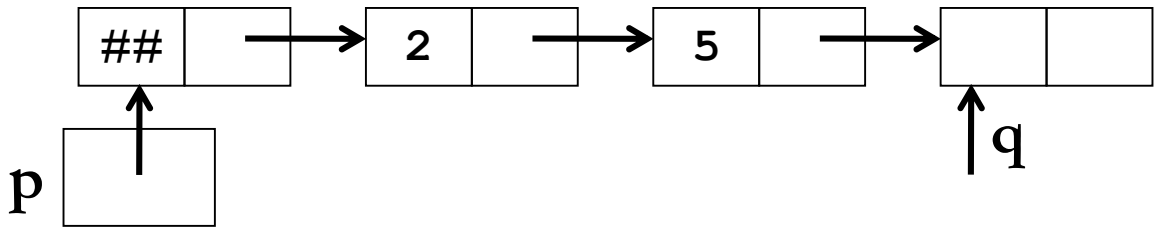
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
        - - - - > q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

/* Formacao e preenchimento do encadeamento de estruturas */

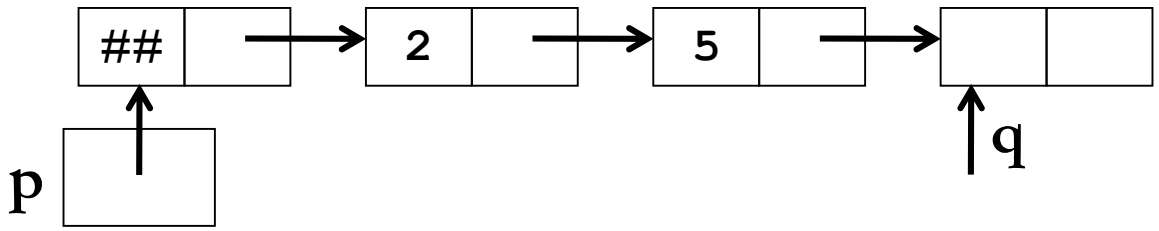
    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
     - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

8

i	3

n	3



```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

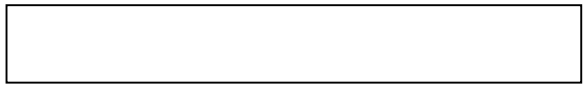
/* Leitura do numero de estruturas */

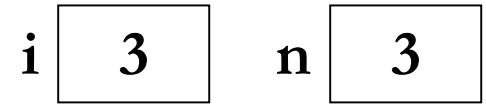
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

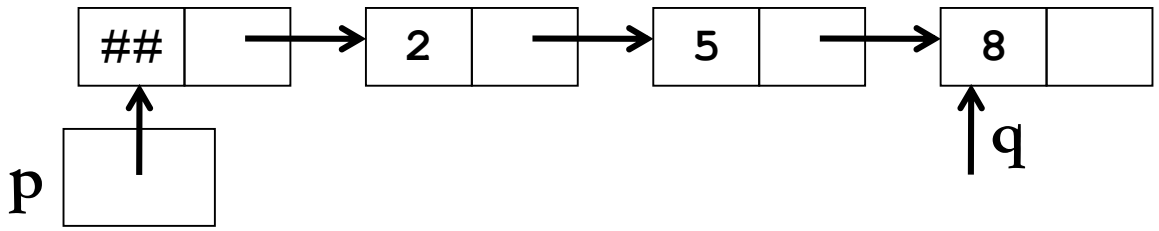
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
     - - - - > scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

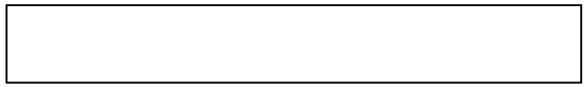
/* Leitura do numero de estruturas */

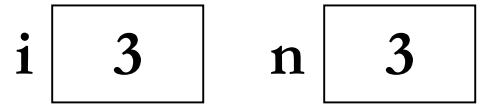
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

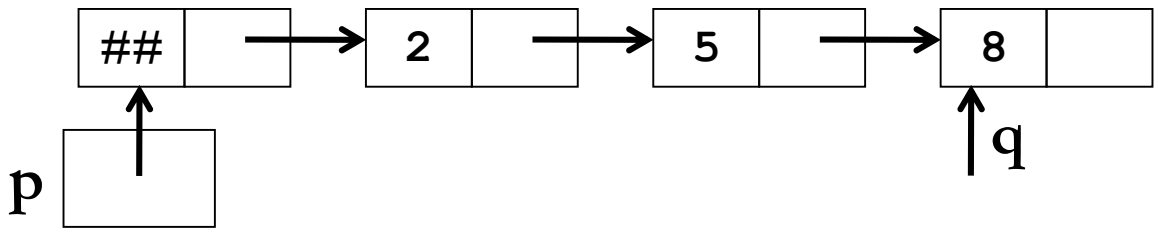
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

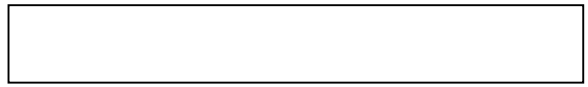
/* Leitura do numero de estruturas */

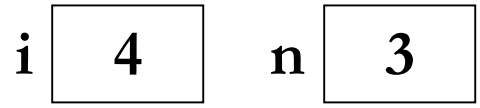
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

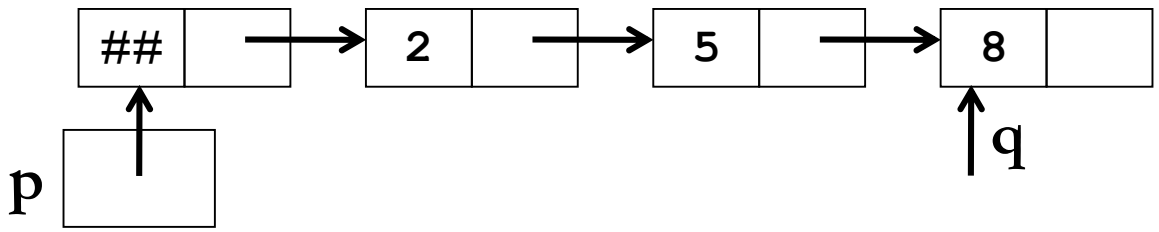
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

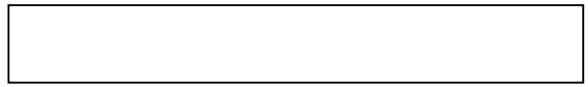
/* Leitura do numero de estruturas */

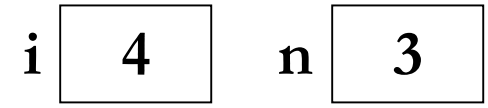
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

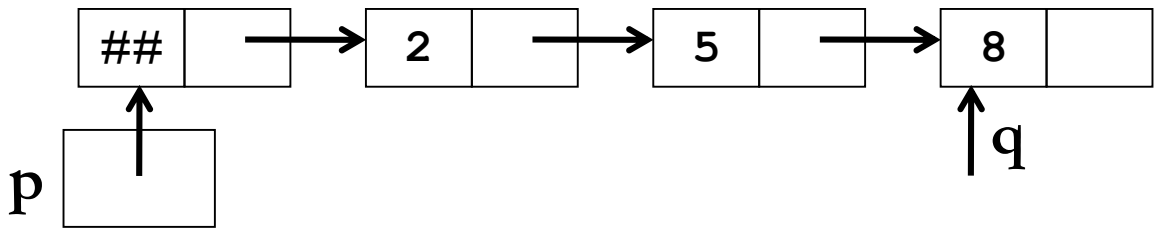
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
        q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

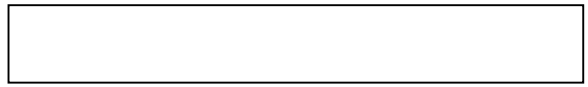
/* Leitura do numero de estruturas */

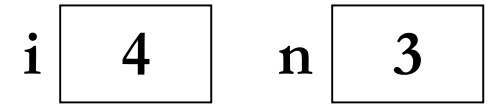
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

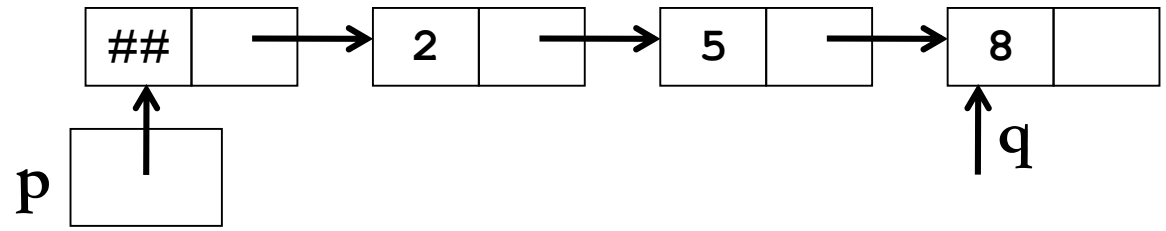
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
- - ->  q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado







```
/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

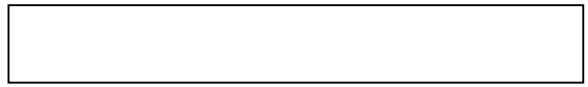
/* Leitura do numero de estruturas */

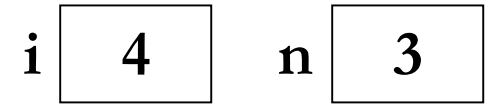
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);

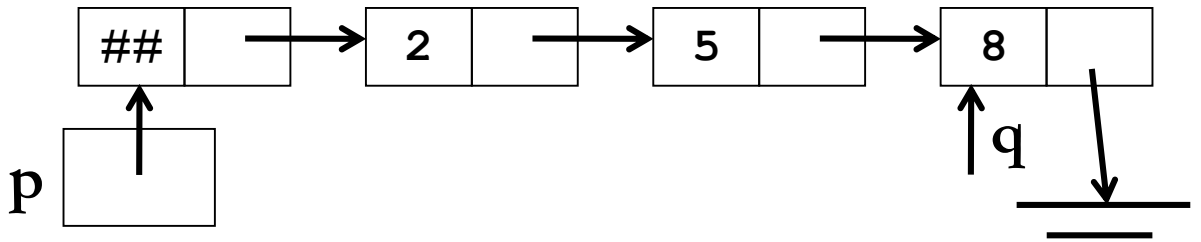
/* Formacao e preenchimento do encadeamento de estruturas */

    if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (q = p, i = 1; i <= n; i++) {
                q->prox = (pontst) malloc (sizeof (st));
                q = q->prox;
                scanf ("%d", &q->a);
        }
- - ->  q->prox = NULL;
   }
   else p->prox = NULL;
```

Buffer do teclado

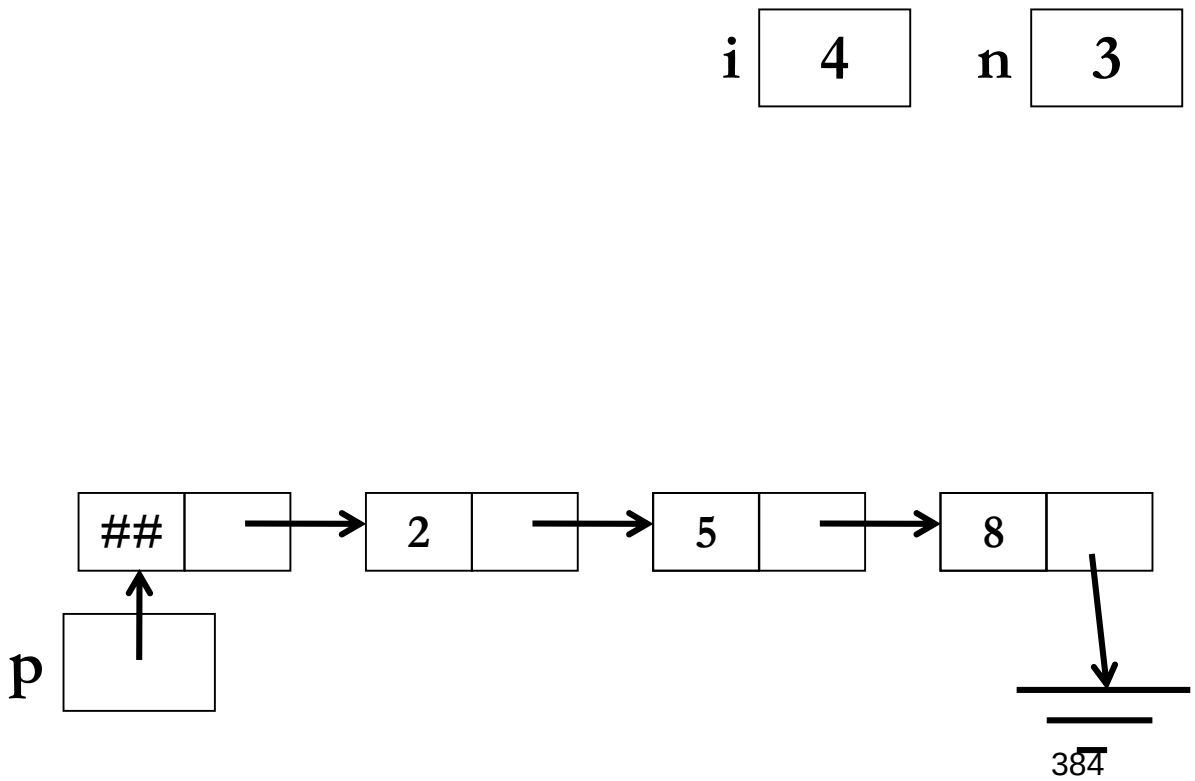






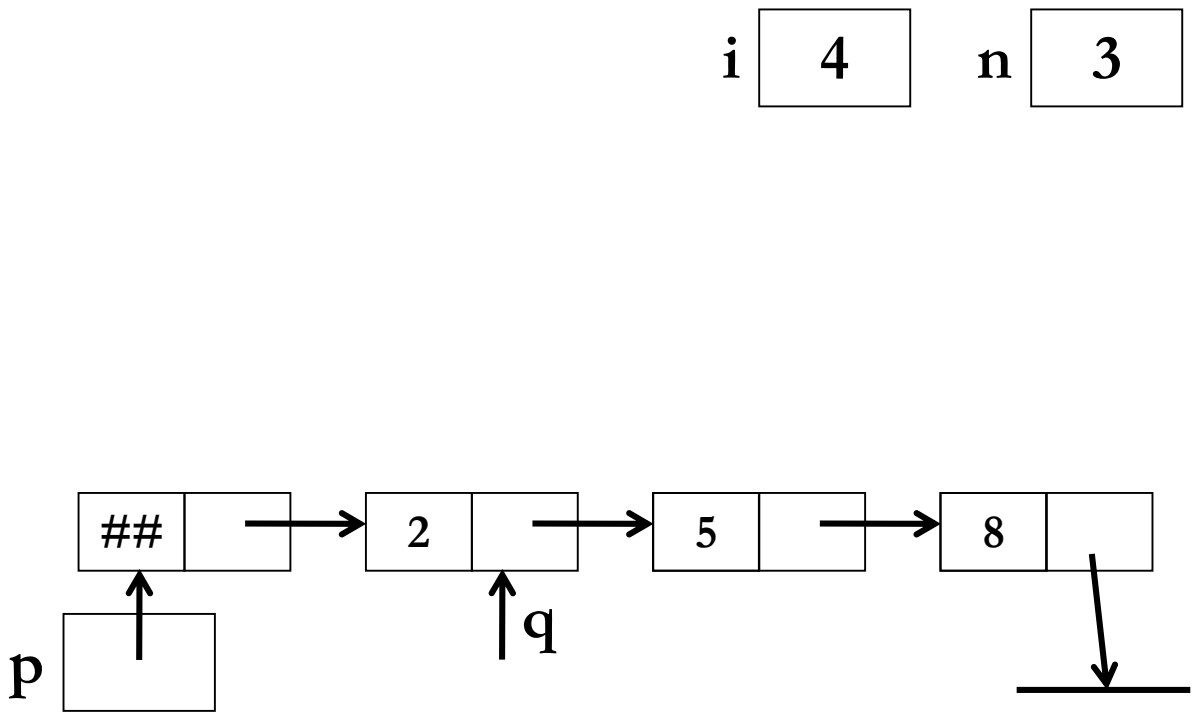
```
/* Escrita do conteudo do encadeamento de estruturas */

   printf ("\n\nConteudo do encadeamento:\n\n\t");
   for (q = p->prox; q != NULL; q = q->prox)
        printf ("%4d", q->a);
```



```
/* Escrita do conteudo do encadeamento de estruturas */

   printf ("\n\nConteudo do encadeamento:\n\n\t");
   for (q = p->prox; q != NULL; q = q->prox)
        printf ("%4d", q->a);
```



```
/* Escrita do conteudo do encadeamento de estruturas */

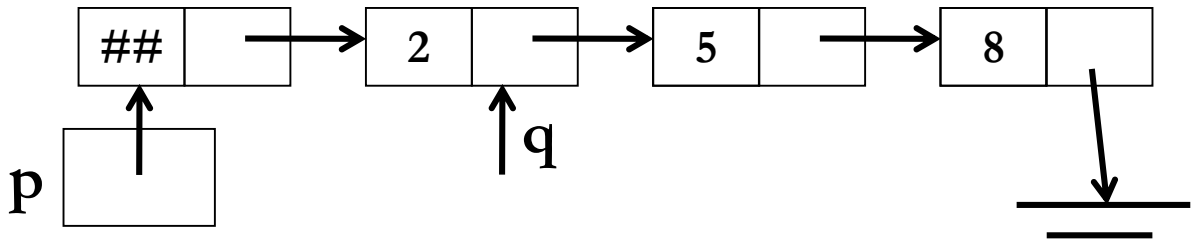
   printf ("\n\nConteudo do encadeamento:\n\n\t");
   for (q = p->prox; q != NULL; q = q->prox)
        printf ("%4d", q->a);

/* Fechamento da tela e encerramento */

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



Copiar, salvar e executar



Programa 11.5: Alterando o **Programa 11.4** para que a formação do encadeamento e a escrita do mesmo fiquem em funções auxiliares

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

/* Declaracoes dos tipos */

typedef struct st st;
struct st {int a; st *prox;};
typedef st *pontst;

/* Declaracoes dos prototipos das funcoes auxiliares */

pontst NovoEncadeamento (void);
void EscreverEncadeamento (pontst);
```

```
/* main: cabecalho e declaracoes locais */

int main () {
   pontst x;

/* Formacao de um encadeamento  */

   printf ("Formacao de um encadeamento:\n\n");
   x = NovoEncadeamento ();

/* Escrita do conteudo do encadeamento formado  */

   printf ("\nConteudo do encadeamento:\n\n\t");
   EscreverEncadeamento (x);

/* Encerramento  */

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

```
/* Funcao NovoEncadeamento: forma um novo encadeamento de
   estruturas, preenchendo-o e retornando um ponteiro para
    a sua estrutura-lider                                    */

pontst NovoEncadeamento () {
   int i, n;
   pontst p, q;

/* Alocacao da estrutura-lider  */

   p = (pontst) malloc (sizeof(st));

/* Leitura do numero de estruturas */

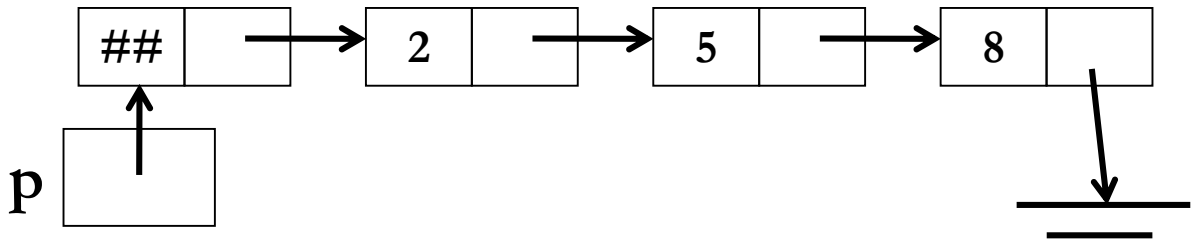
   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);
```

```
/* Formacao e preenchimento do encadeamento de estruturas */

   if (n > 0) {
       printf ("\nDigite os elementos: ");
       for (q = p, i = 1; i <= n; i++) {
              q->prox = (pontst) malloc (sizeof (st));
              q = q->prox;
              scanf ("%d", &q->a);
       }
       q->prox = NULL;
   }
   else p->prox = NULL;
```

```
/* Formacao e preenchimento do encadeamento de estruturas */

   if (n > 0) {
       printf ("\nDigite os elementos: ");
       for (q = p, i = 1; i <= n; i++) {
              q->prox = (pontst) malloc (sizeof (st));
              q = q->prox;
              scanf ("%d", &q->a);
       }
       q->prox = NULL;
   }
   else p->prox = NULL;
```

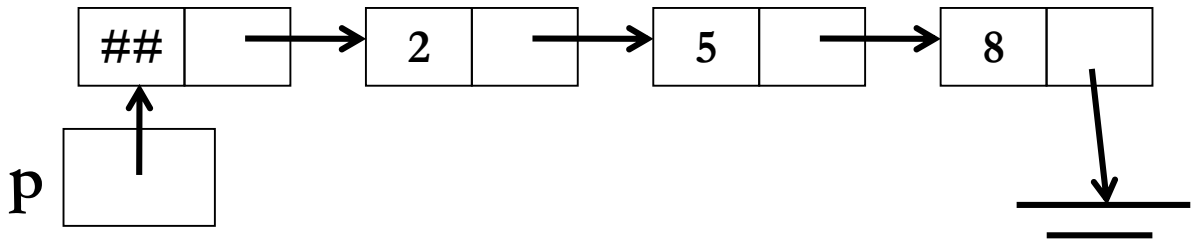


```
/* Formacao e preenchimento do encadeamento de estruturas */

   if (n > 0) {
       printf ("\nDigite os elementos: ");
       for (q = p, i = 1; i <= n; i++) {
              q->prox = (pontst) malloc (sizeof (st));
              q = q->prox;
              scanf ("%d", &q->a);
       }
       q->prox = NULL;
   }
   else p->prox = NULL;

/* Retornando o ponteiro para a estrutura-lider */

   return p;
}
```



```
/* Formacao e preenchimento do encadeamento de estruturas */

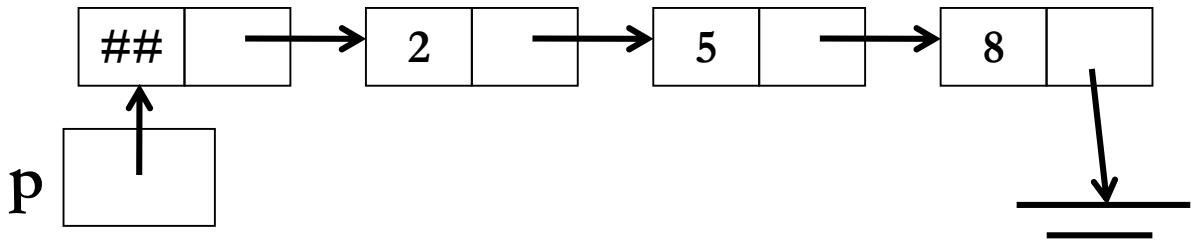
   if (n > 0) {
       printf ("\nDigite os elementos: ");
       for (q = p, i = 1; i <= n; i++) {
              q->prox = (pontst) malloc (sizeof (st));
              q = q->prox;
              scanf ("%d", &q->a);
       }
       q->prox = NULL;
   }
   else p->prox = NULL;

/* Retornando o ponteiro para a estrutura-lider */

   return p;
}
```

Na função **main**:

x = NovoEncadeamento ();



```
/* Formacao e preenchimento do encadeamento de estruturas */

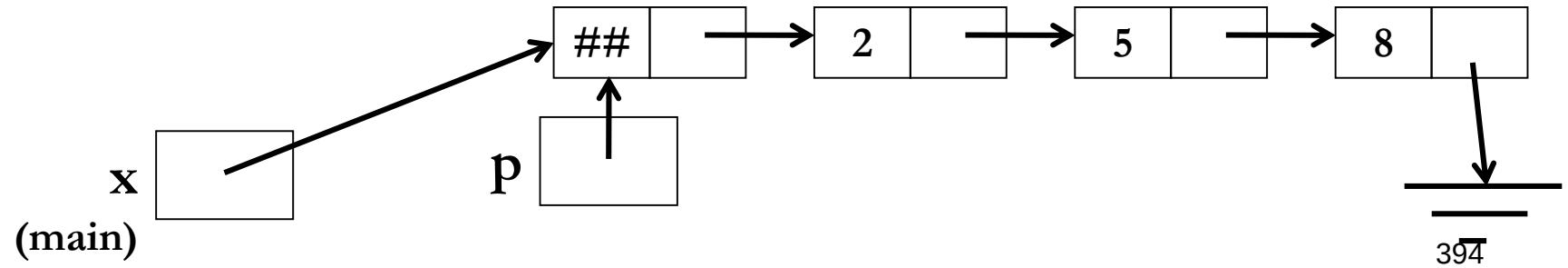
   if (n > 0) {
       printf ("\nDigite os elementos: ");
       for (q = p, i = 1; i <= n; i++) {
              q->prox = (pontst) malloc (sizeof (st));
              q = q->prox;
              scanf ("%d", &q->a);
       }
       q->prox = NULL;
   }
   else p->prox = NULL;

/* Retornando o ponteiro para a estrutura-lider */

   return p;
}
```

Na função **main**:

x = NovoEncadeamento ();



```
/* Funcao EscreverEncadeamento: escreve o conteudo
   do encadeamento de estruturas acoplado ao parametro p */

void EscreverEncadeamento (pontst p){
   pontst q;

   for (q = p->prox; q != NULL; q = q->prox)
       printf ("%4d", q->a);
}
```

```
/* Funcao EscreverEncadeamento: escreve o conteudo
   do encadeamento de estruturas acoplado ao parametro p */

void EscreverEncadeamento (pontst p){
   pontst q;

   for (q = p->prox; q != NULL; q = q->prox)
       printf ("%4d", q->a);
}
```

Na função **main**:

EscreverEncadeamento (x);

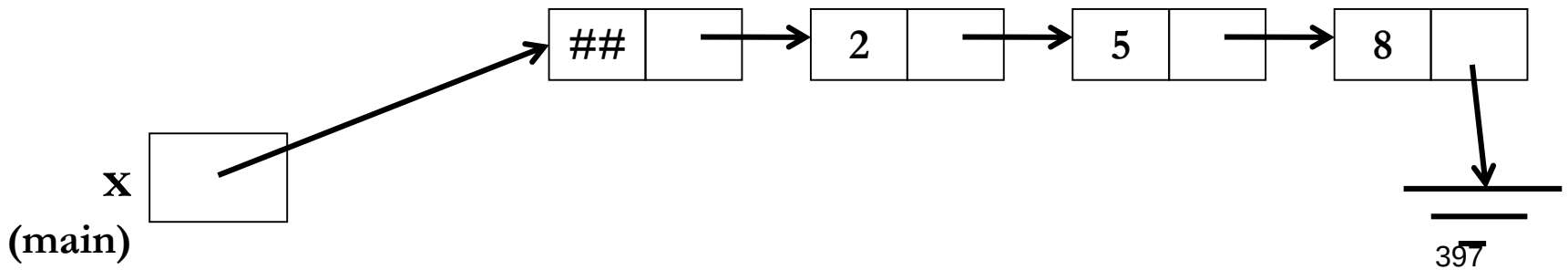
```
/* Funcao EscreverEncadeamento: escreve o conteudo
   do encadeamento de estruturas acoplado ao parametro p */

void EscreverEncadeamento (pontst p){
   pontst q;

   for (q = p->prox; q != NULL; q = q->prox)
        printf ("%4d", q->a);
}
```

Na função **main**:

EscreverEncadeamento (x);



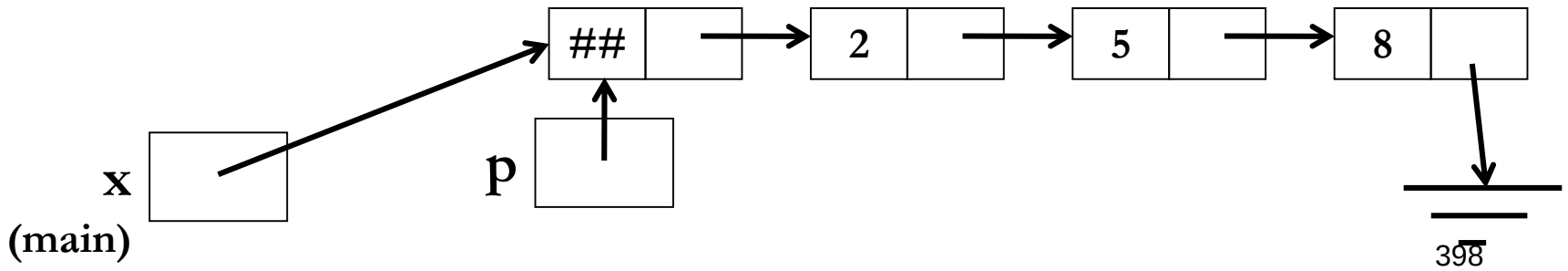
```
/* Funcao EscreverEncadeamento: escreve o conteudo
   do encadeamento de estruturas acoplado ao parametro p */

void EscreverEncadeamento (pontst p){
   pontst q;

   for (q = p->prox; q != NULL; q = q->prox)
        printf ("%4d", q->a);
}
```

Na função **main**:

EscreverEncadeamento (x);



```
/* Funcao EscreverEncadeamento: escreve o conteudo
   do encadeamento de estruturas acoplado ao parametro p */

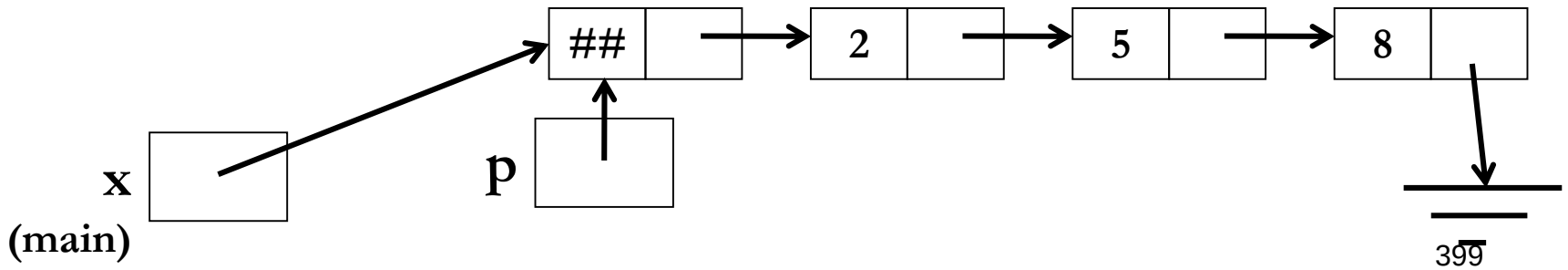
void EscreverEncadeamento (pontst p){
   pontst q;

   for (q = p->prox; q != NULL; q = q->prox)
        printf ("%4d", q->a);
}
```

Na função **main**:

EscreverEncadeamento (x);

Copiar, salvar e executar



Programa 11.6: Alterando a função **NovoEncadeamento** do **Programa 11.5** para que cada número seja inserido mantendo ordenação crescente

```
pontst NovoEncadeamento () {
   int i, n, num;
   pontst p, q, r;

/* Alocacao da estrutura-lider, aterrando-a  */

   p = (pontst) malloc (sizeof(st));
   p->prox = NULL;

/* Leitura do numero de estruturas */

   printf ("Digite o numero de elementos do encadeamento: ");
   scanf ("%d", &n);
```

```
/* Formacao e preenchimento do encadeamento de estruturas */

   if (n > 0) {
        printf ("\nDigite os elementos: ");
        for (i = 1; i <= n; i++) {
               scanf ("%d", &num);
               q = p;
               while (q->prox != NULL && q->prox->a < num)
                      q = q->prox;
               r = q->prox;
               q->prox = (pontst) malloc (sizeof (st));
               q->prox->a = num;
               q->prox->prox = r;
        }
   }

/* Retornando o ponteiro para a estrutura-lider */

   return p;
}
```

Copiar, salvar e executar

11º Exercício Laboratorial: Armazenamento e manipulação de números inteiros muito grandes, em encadeamentos de estruturas

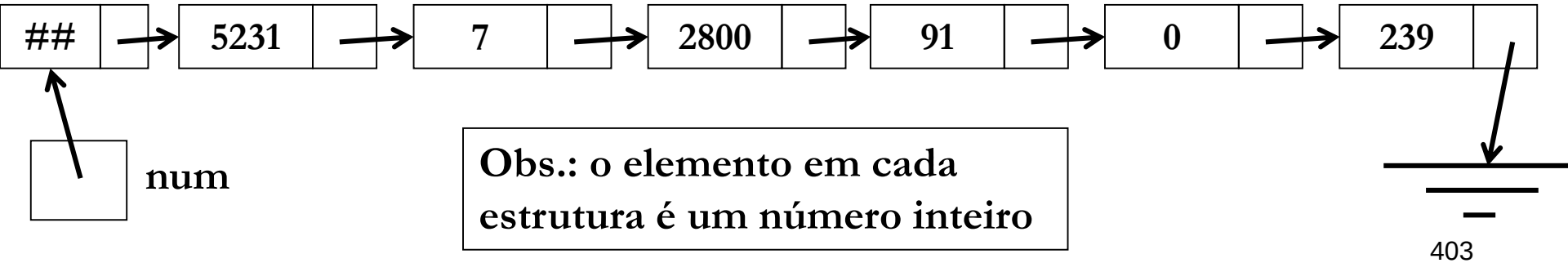
- Números inteiros em C devem caber em **4 bytes**
- Os comandos de C não conseguem manipular números inteiros **muito grandes** (> 10 bilhões)
- Pode-se armazená-los cada um num **encadeamento de estruturas** e assim fazer somas, multiplicações, etc.
- A finalidade do exercício é **ler dois números** (do teclado), **armazená**-los em encadeamentos de estruturas, **somá**-los, **multiplicá**-los e **escrever** os resultados (no vídeo)

1º Detalhe: Esquema de armazenamento

- Seja o seguinte número: **239000000912800000075231**
- Dividindo-o em **grupos de 4 dígitos**, da direita para a esquerda:

239-0000-0091-2800-0007-5231

- Armazenando os **grupos**, cada um numa **estrutura**, e encadeando-os:

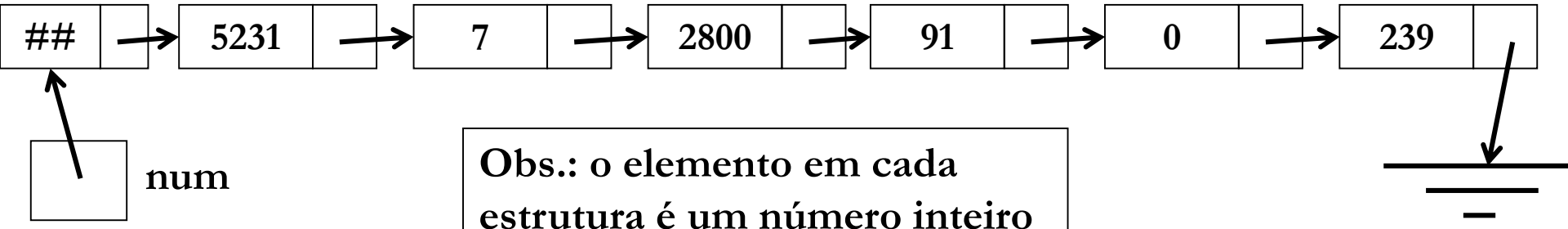


Obs.: o elemento em cada estrutura é um número inteiro

1º Detalhe: Esquema de armazenamento

- **Possíveis declarações:**

```
typedef struct celula celula;
typedef celula* pont;
struct celula {
    int num;
    pont prox;
};
typedef pont numero;
```



Obs.: o elemento em cada estrutura é um número inteiro

2º Detalhe: Leitura e armazenamento

- Ler o número como **cadeia de caracteres**:

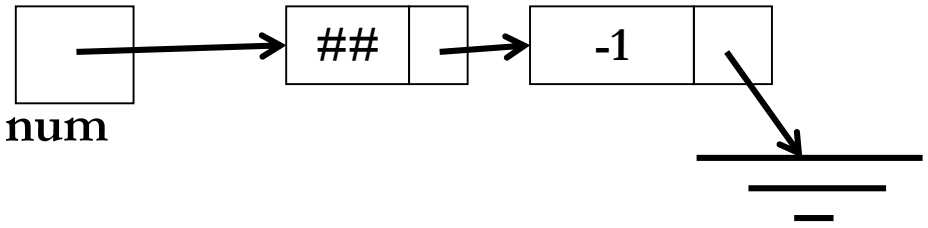
  “239000000912800000075231”

- Armazená-lo inicialmente numa **variável** cadeia de caracteres:

'2'	'3'	'9'	'0'	'0'	'0'	'0'	'0'	'0'	'9'	'1'	'2'	'8'	'0'	'0'	'0'	'0'	'0'	'7'	'5'	'2'	'3'	'1'	'\0'

cad

- Checar a existência de caracteres **não-dígitos**: caso haja, armazenar só **-1** no encadeamento, indicando irregularidade



2º Detalhe: Leitura e armazenamento

- **Eliminar** da variável cadeia os eventuais **zeros à esquerda**

- Por exemplo, se for digitado:

  "000000912800000075231"

- Armazenando na cadeia, fica:

'0'	'0'	'0'	'0'	'0'	'0'	'9'	'1'	'2'	'8'	'0'	'0'	'0'	'0'	'0'	'7'	'5'	'2'	'3'	'1'	'\0'

cad

- Eliminando os **zeros à esquerda**:

'9'	'1'	'2'	'8'	'0'	'0'	'0'	'0'	'0'	'7'	'5'	'2'	'3'	'1'	'\0'

cad

- O **único zero à esquerda** que não deve se eliminado é aquele que aparecer imediatamente **antes do '\0'**

2º Detalhe: Leitura e armazenamento

- Para armazenar o número no encadeamento, deve-se **dividir** a cadeia em **grupos** de no máximo **4 caracteres**

- Por exemplo, se a cadeia for:

"23900000091280000075231"

- Ela será dividida nas cadeias:

"239", "0000", "0091", "2800", "0007", "5231"

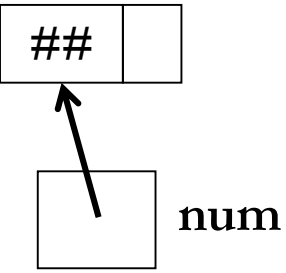
- Usando a função **atoi**, converte-se cada cadeia no inteiro correspondente:

239, 0, 91, 2800, 7, 5231

2º Detalhe: Leitura e armazenamento

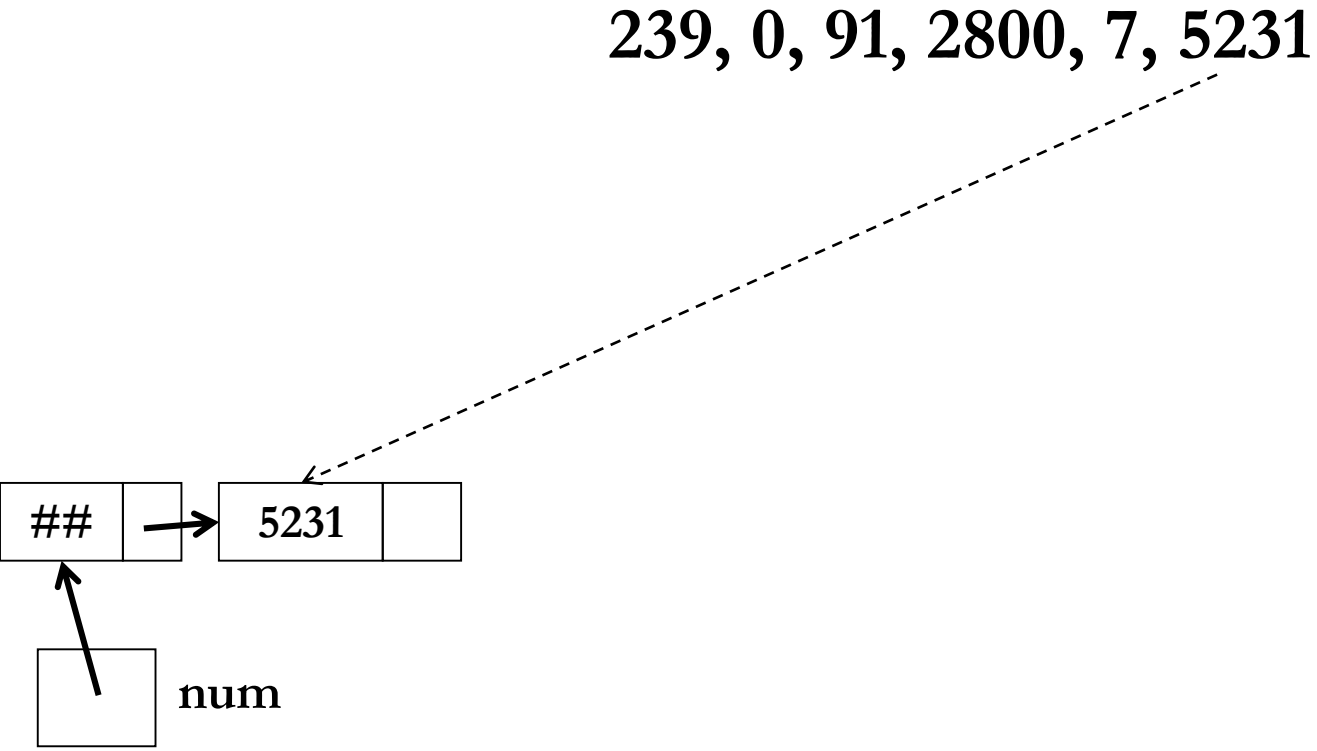
- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:

239, 0, 91, 2800, 7, 5231



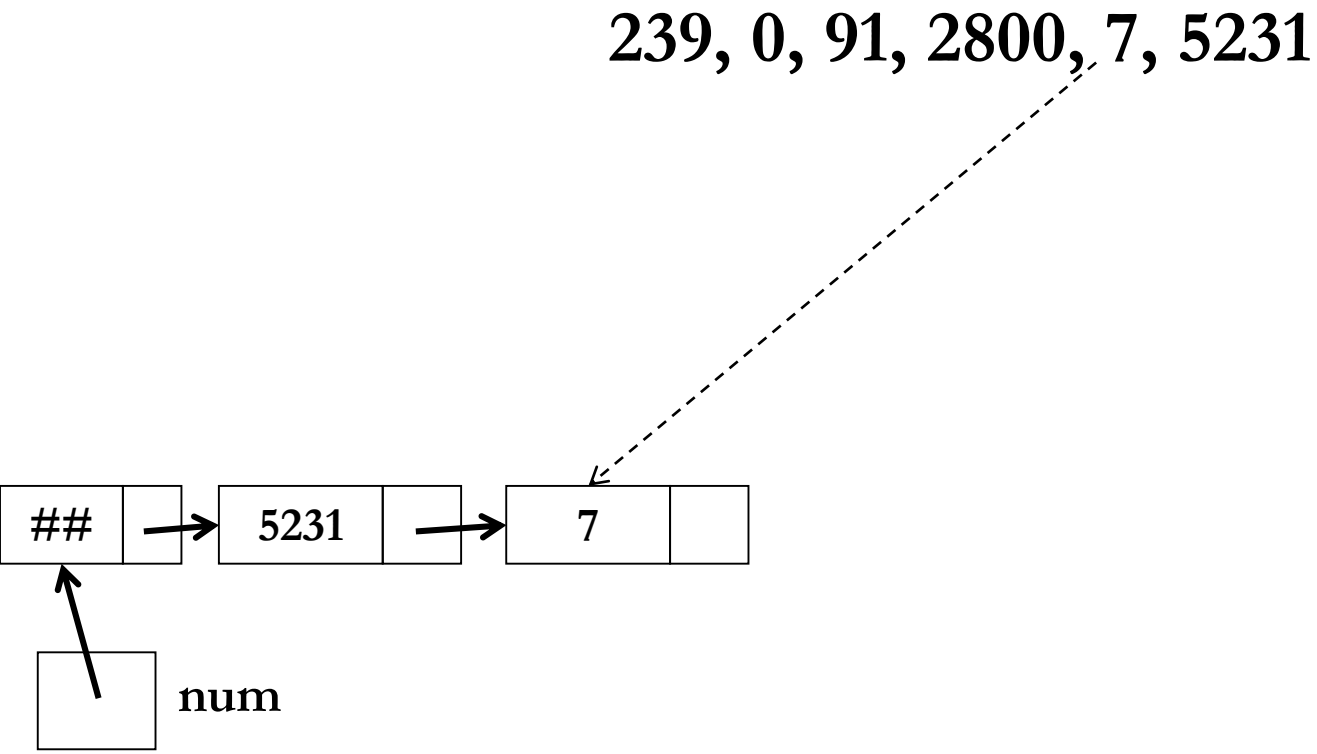
2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:



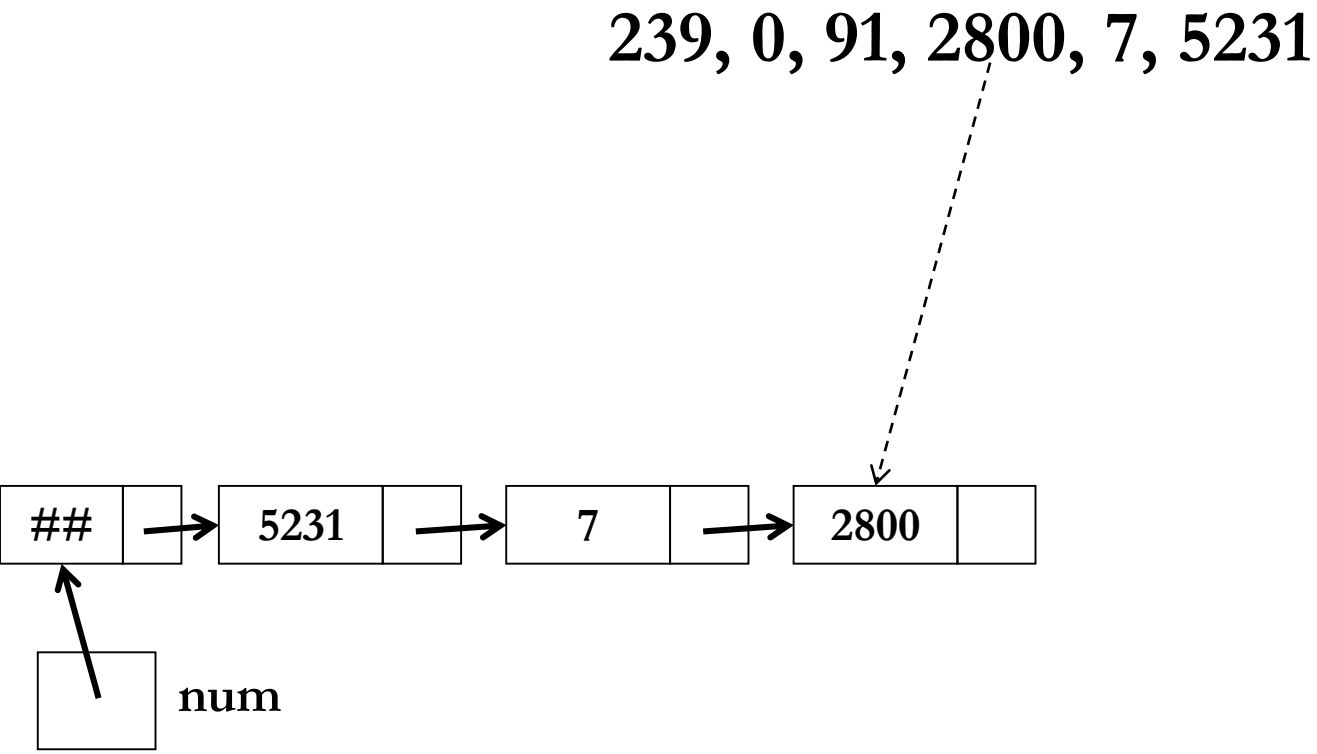
2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:



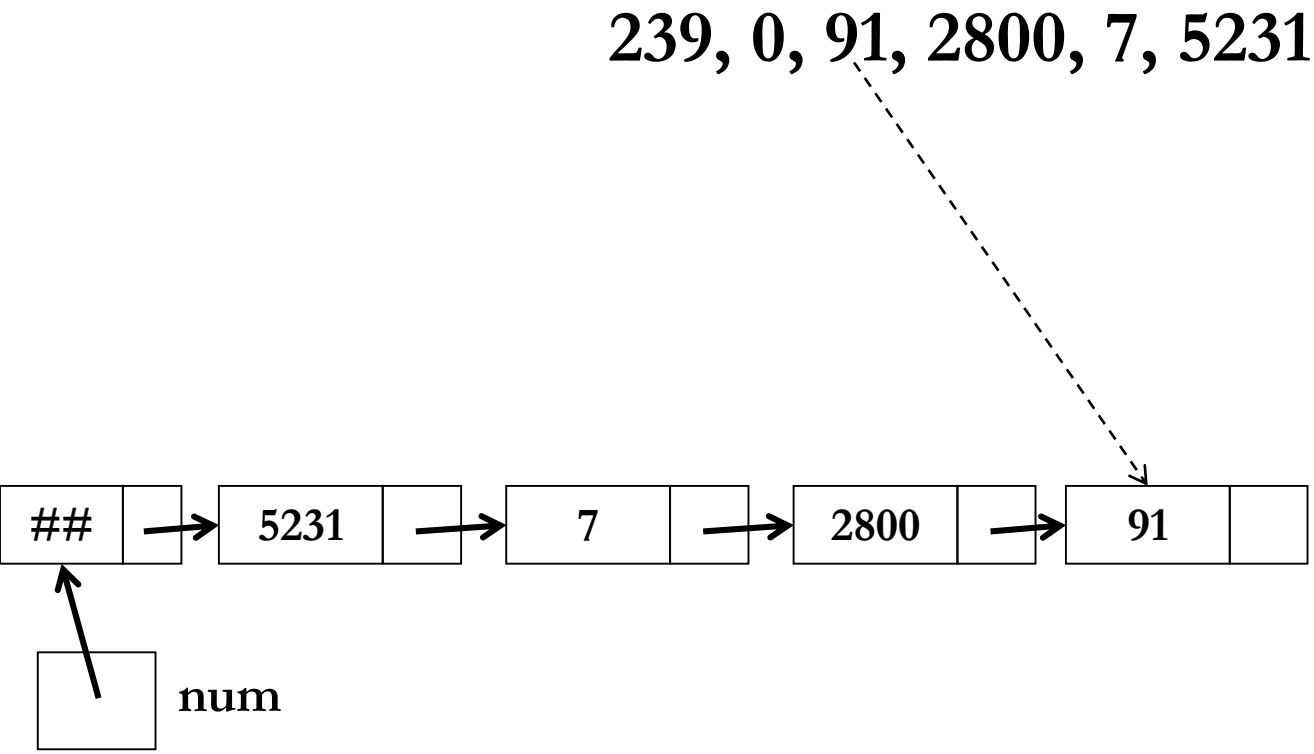
2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:



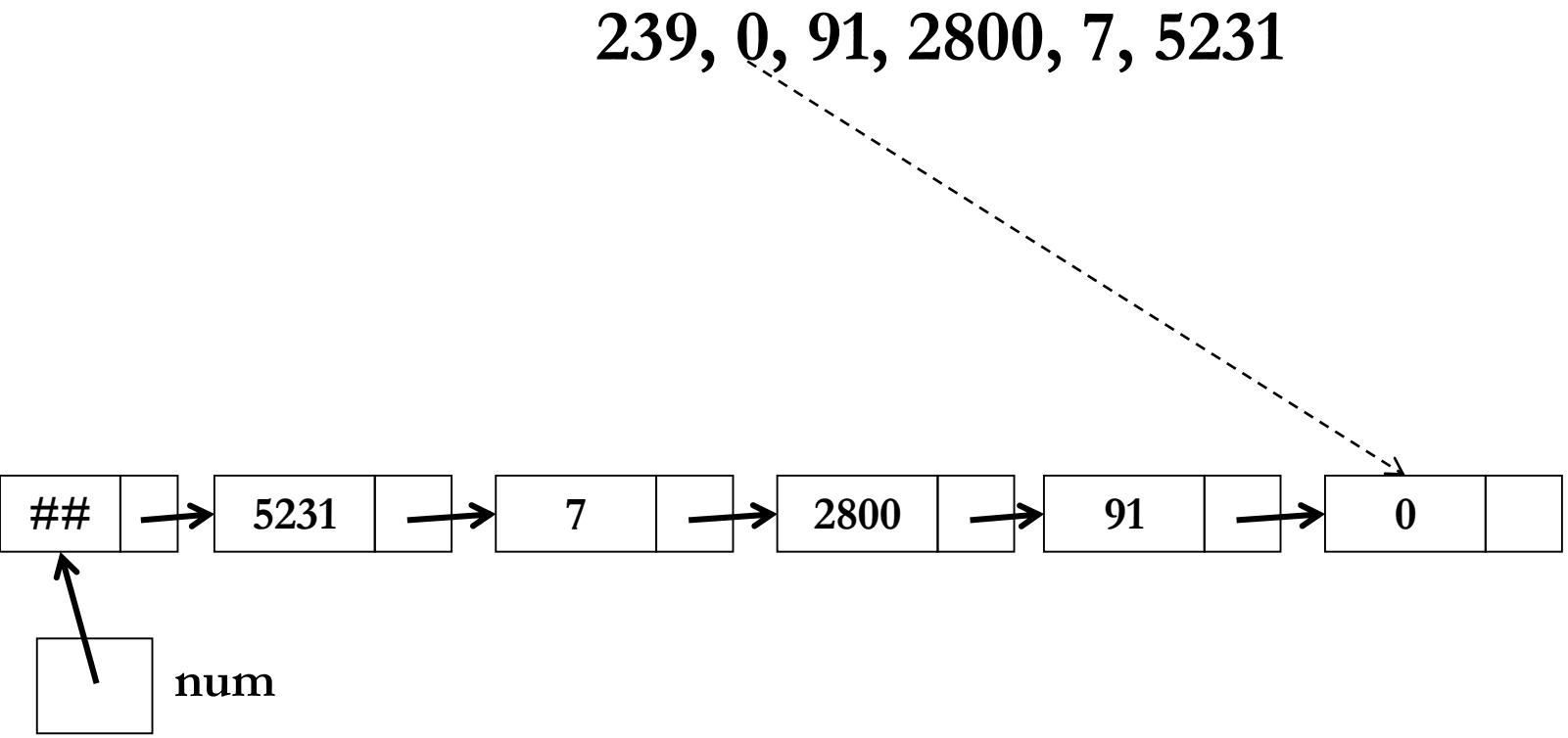
2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:



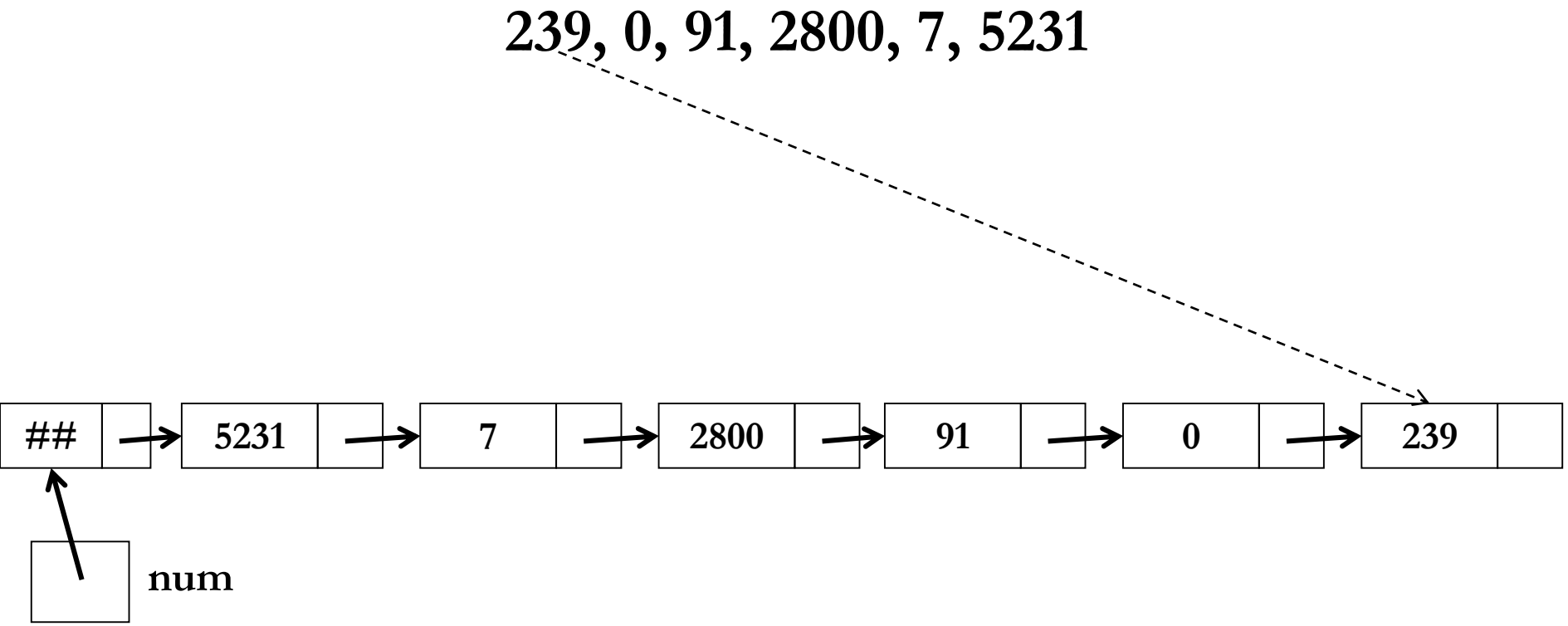
2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:



2º Detalhe: Leitura e armazenamento

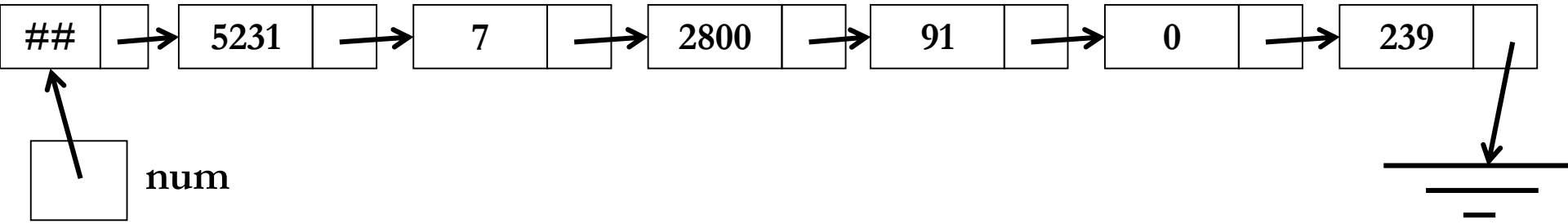
- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:



2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:

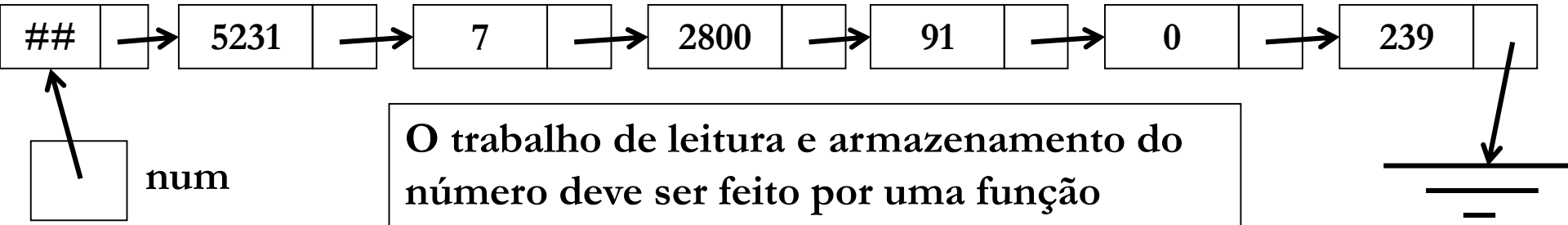
239, 0, 91, 2800, 7, 5231



2º Detalhe: Leitura e armazenamento

- Então, forma-se o encadeamento, armazenando nele cada um dos inteiros obtidos:

239, 0, 91, 2800, 7, 5231



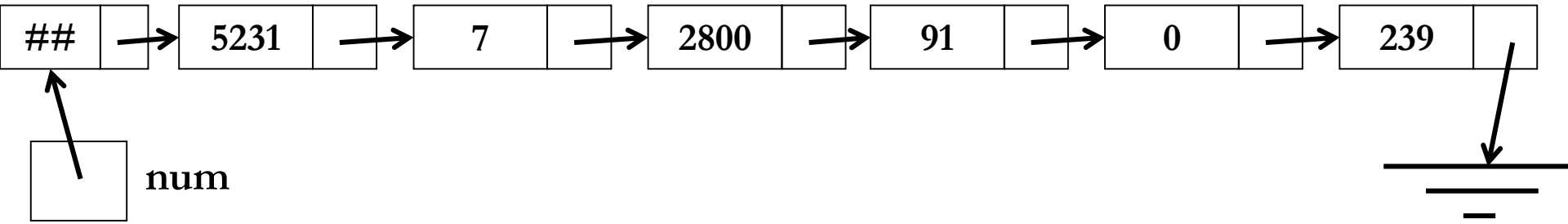
O trabalho de leitura e armazenamento do número deve ser feito por uma função

A função deve retornar o ponteiro num, tal como no Programa 11.5

3º Detalhe: Escrita do conteúdo do encadeamento

- Para o número ser **escrito**, o conteúdo de cada estrutura deve ser convertido para **cadeia de caracteres**:

- Por exemplo, o encadeamento:



deve produzir as seguintes cadeias:

“5231”, “7”, “2800”, “91”, “0”, “239”

3º Detalhe: Escrita do conteúdo do encadeamento

“5231”, “7”, “2800”, “91”, “0”, “239”

- Com a exceção da última, todas as cadeias devem ter 4 caracteres (completar com zeros à esquerda):

“5231”, “0007”, “2800”, “0091”, “0000”, “239”

- Estas cadeias devem ser **concatenadas** da direita para a esquerda e, em seguida, a concatenação pode ser **escrita**:

“2390000009128000075231”

O trabalho de escrita do encadeamento deve ser feito por uma função

A função deve ser do tipo void e deve ter como parâmetro o ponteiro para a estrutura-líder do encadeamento, tal como no Programa 11.5

4º Detalhe: Soma dos números de dois encadeamentos

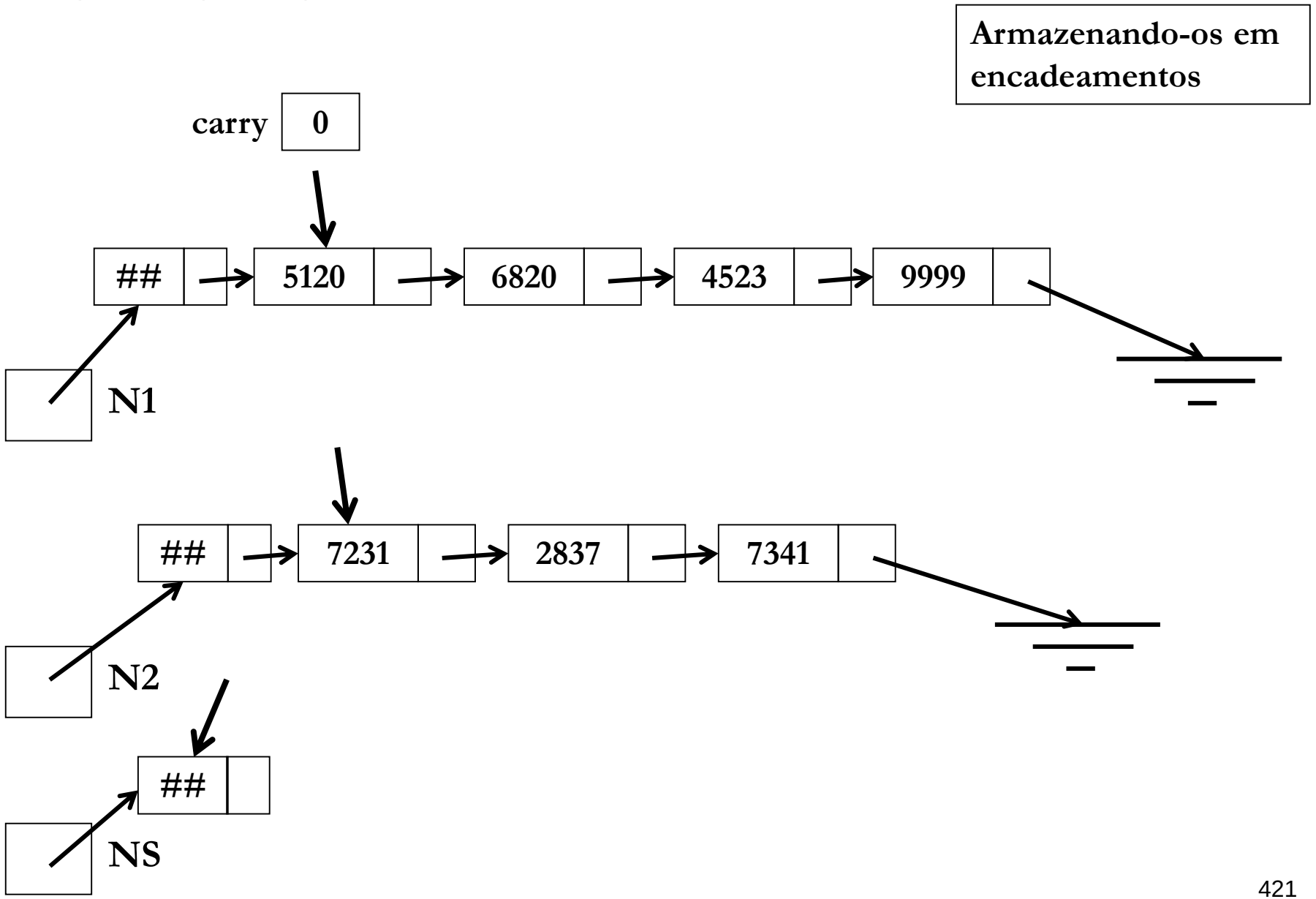
- Fazer uma **função** para:
  - Receber como argumentos **dois encadeamentos** contendo números inteiros, conforme descrito anteriormente
  - Produzir um **encadeamento** contendo o resultado da **soma** dos números recebidos como argumentos

4º Detalhe: Soma dos números de dois encadeamentos

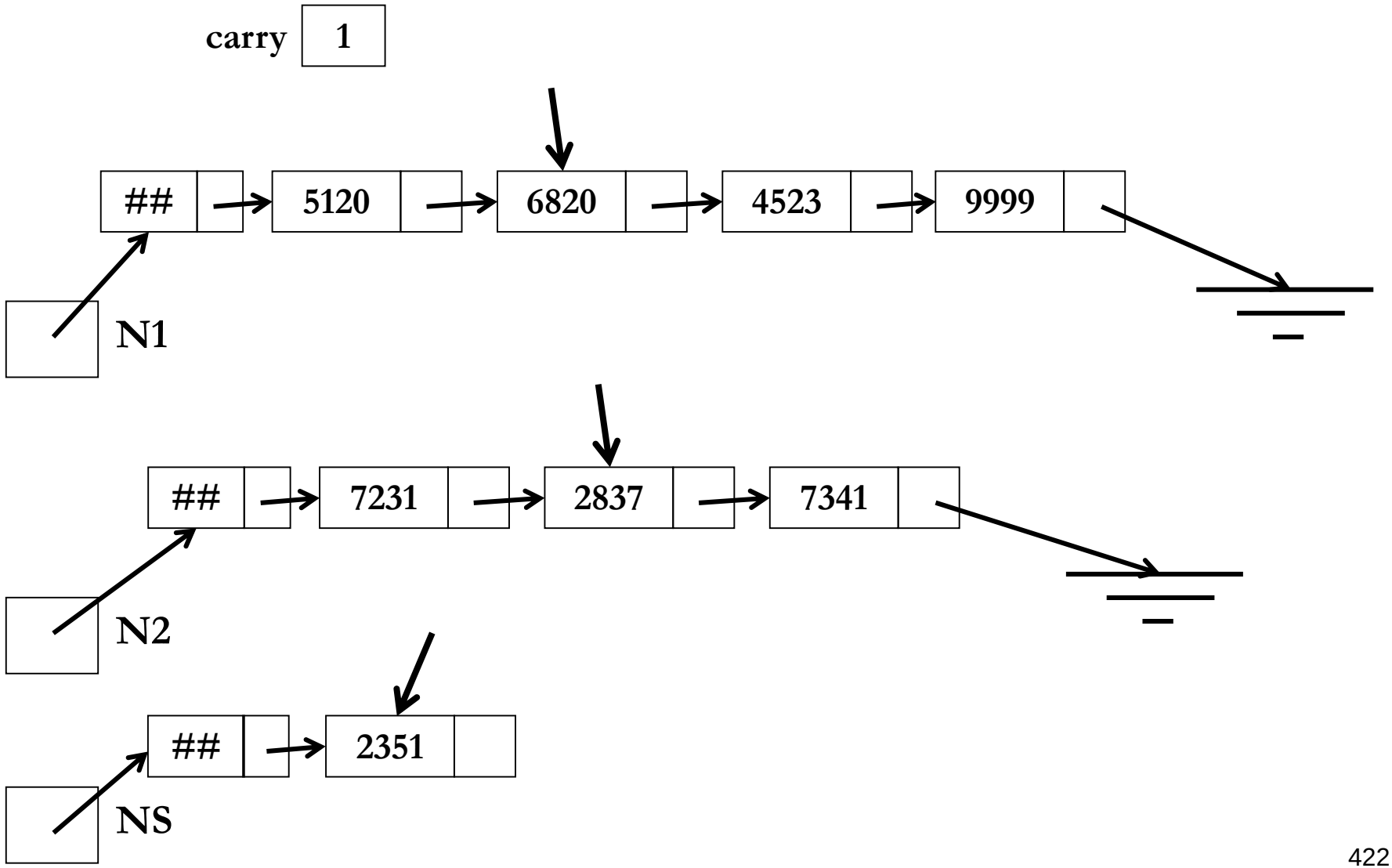
- Caso pelo menos um dos encadeamentos guarde um **número irregular**, o resultado também deve guardar um **número irregular** semelhante

- A soma deve ser calculada **percorrendo**-se os dois encadeamentos recebidos como argumentos, para **alocar** e **preencher** cada estrutura do encadeamento do **resultado**

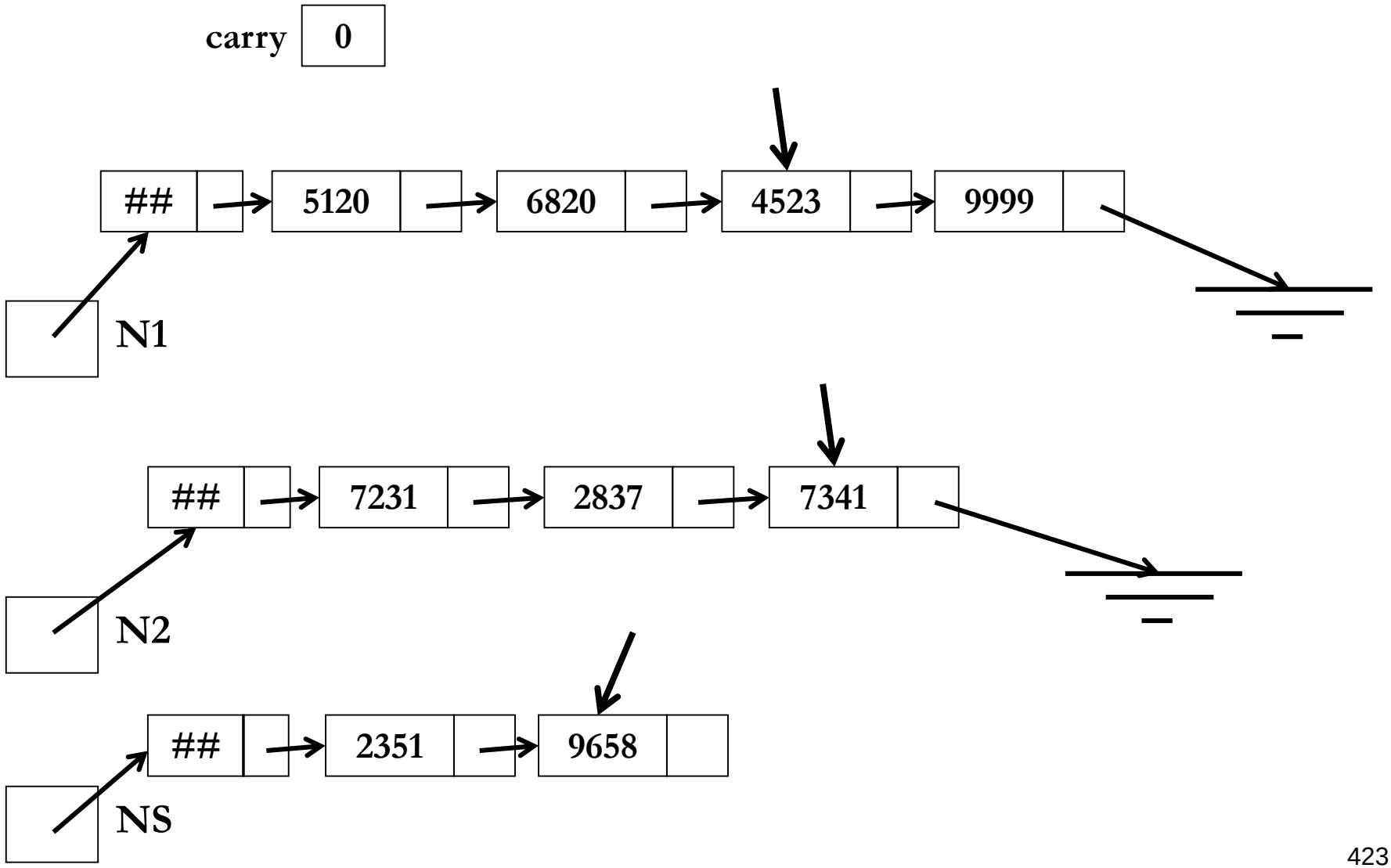
Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



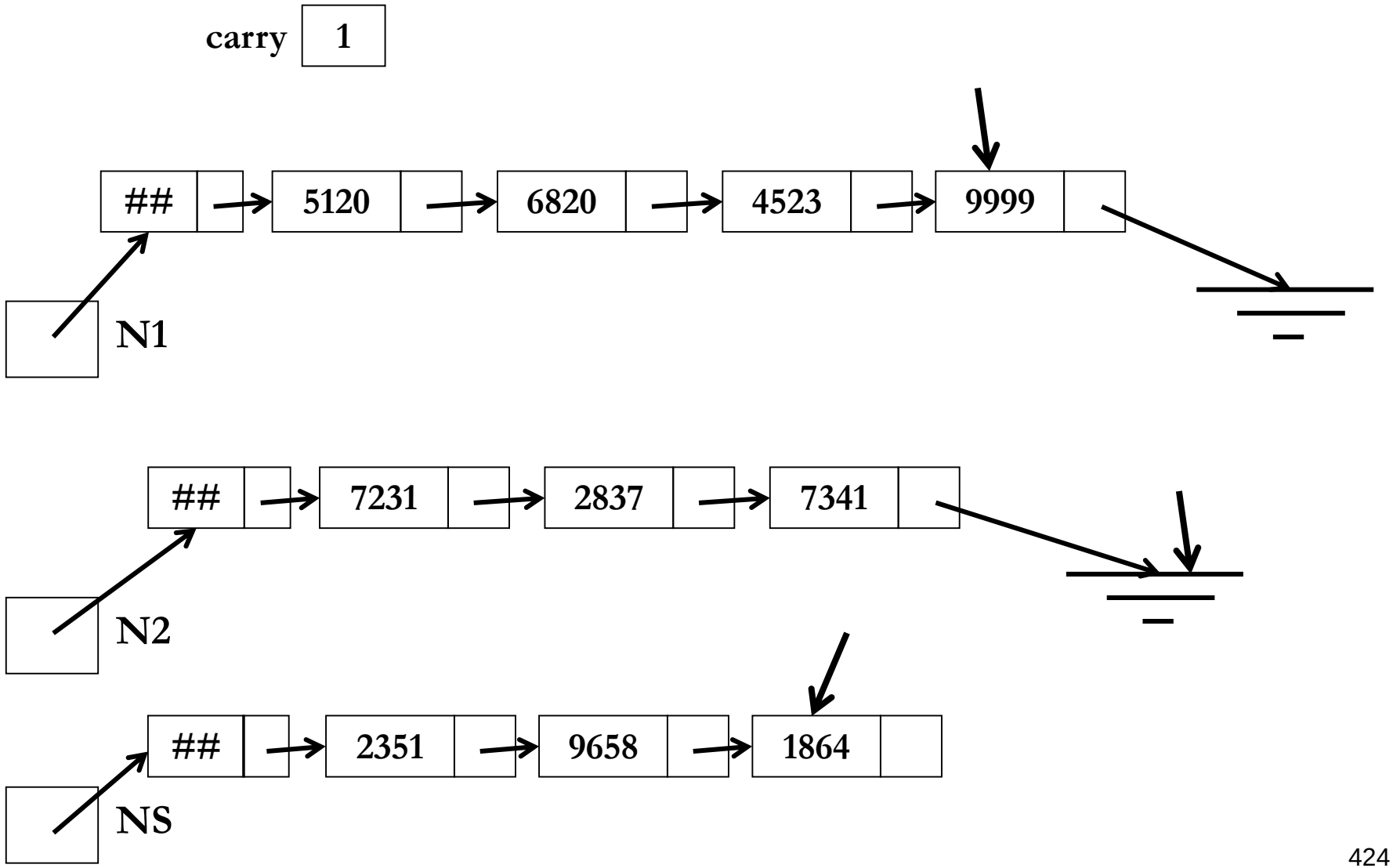
Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



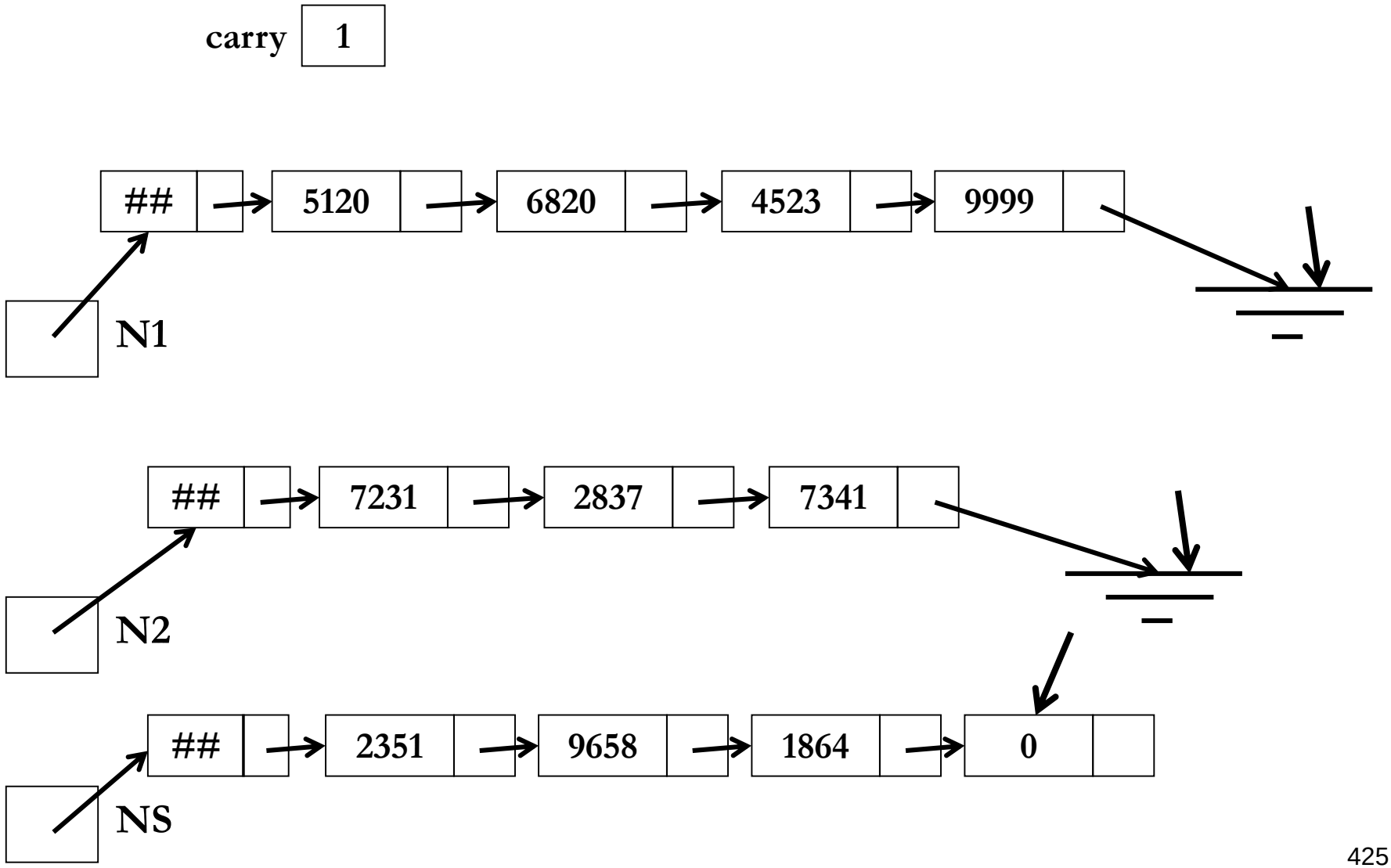
Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



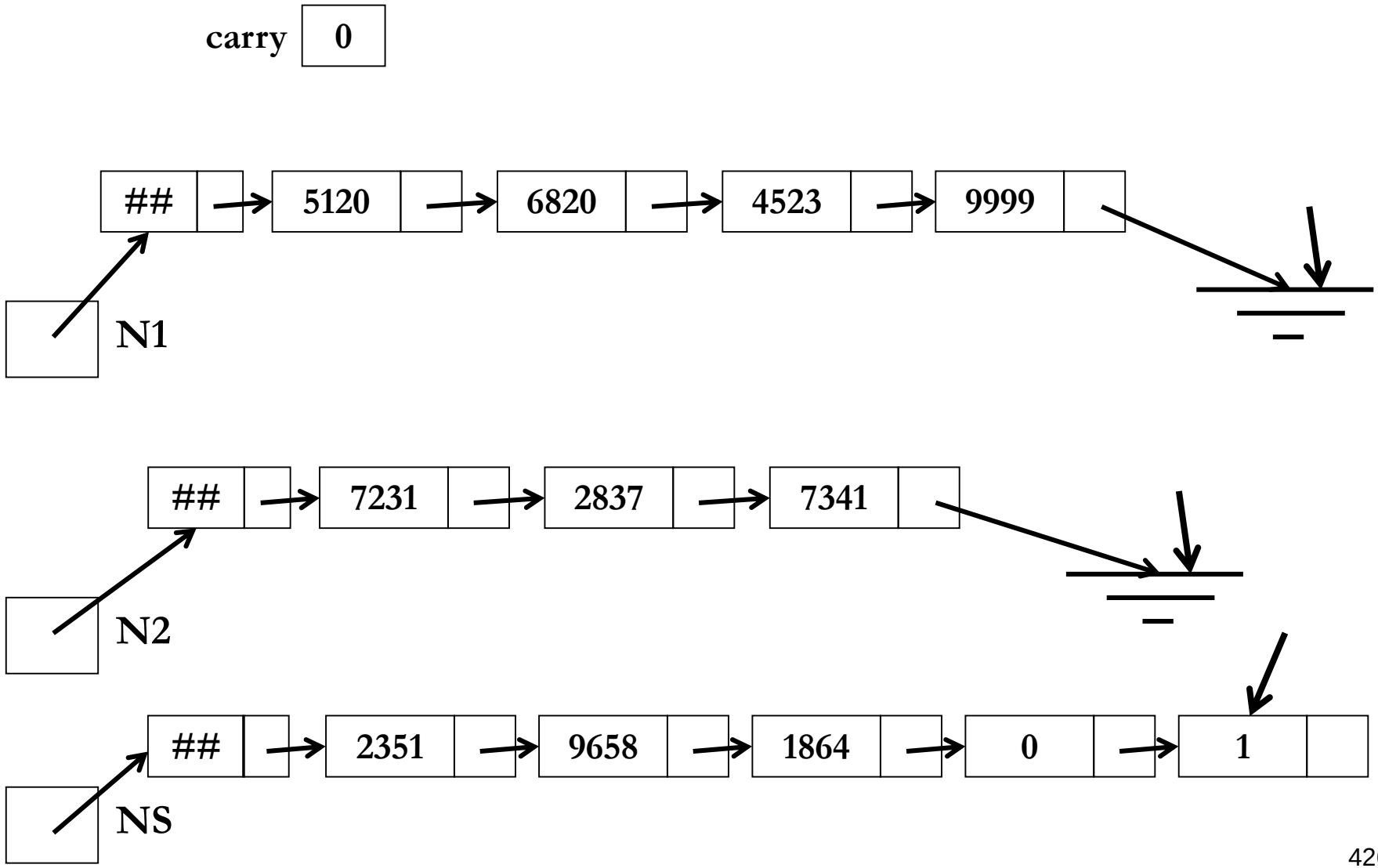
Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



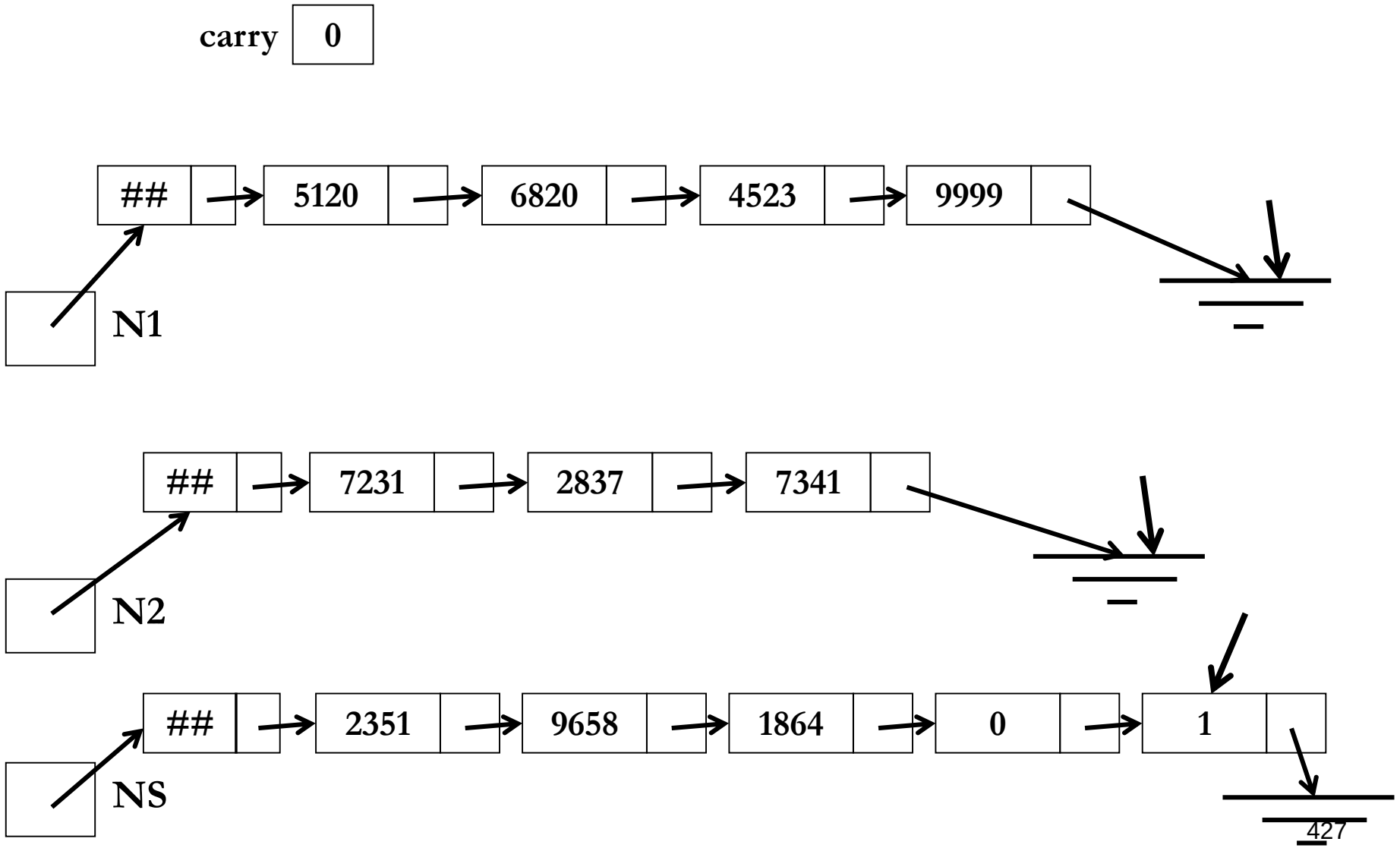
Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



Exemplo: Seja a soma dos números **9999452368205120** e **734128377231**



5º Detalhe: Multiplicação dos números de dois encadeamentos

- Fazer uma função para:
  - Receber como argumentos **dois encadeamentos** contendo números inteiros, conforme descrito anteriormente
  - Produzir um encadeamento contendo o resultado da **multiplicação** dos números recebidos como argumentos

5º Detalhe: Multiplicação dos números de dois encadeamentos

- Caso pelo menos um dos encadeamentos guarde um **número irregular**, o resultado também deve guardar um **número irregular** semelhante

- A multiplicação deve fazer **sucessivas chamadas** da função para **somar** dois números armazenados em encadeamentos de estruturas

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	5120
7341	0000	2837	7231

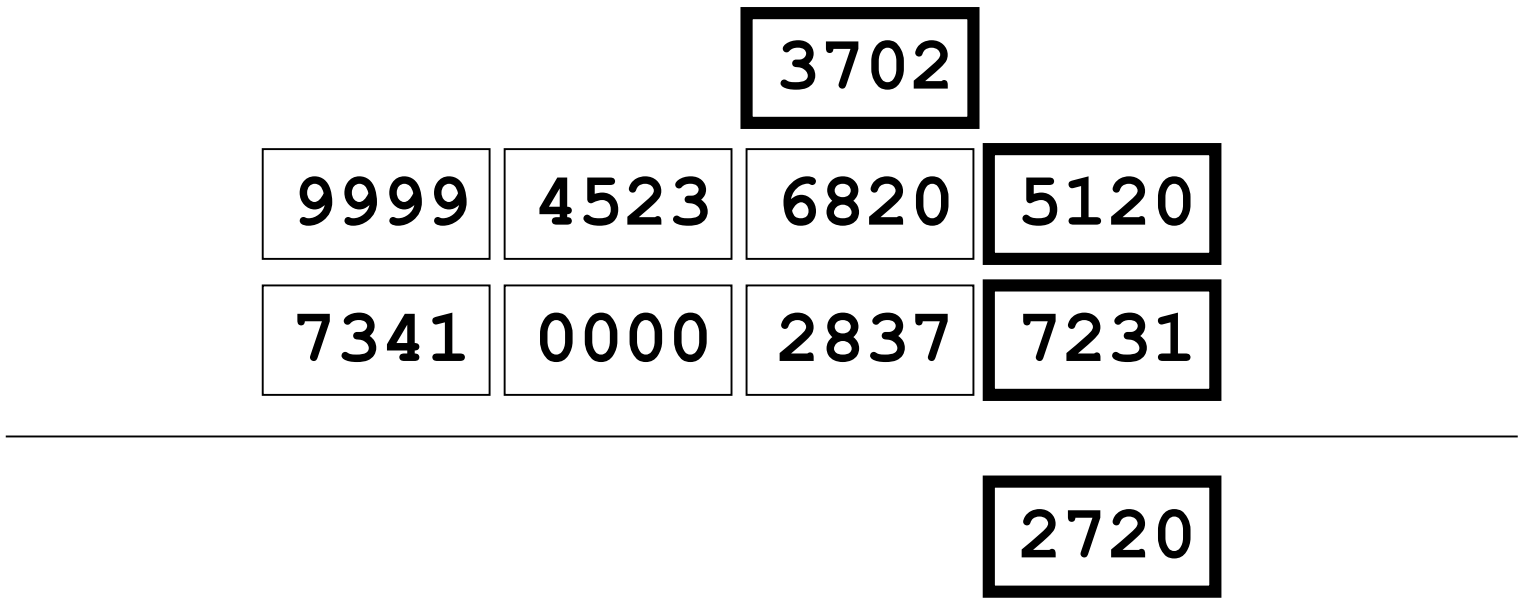
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	5120
7341	0000	2837	7231

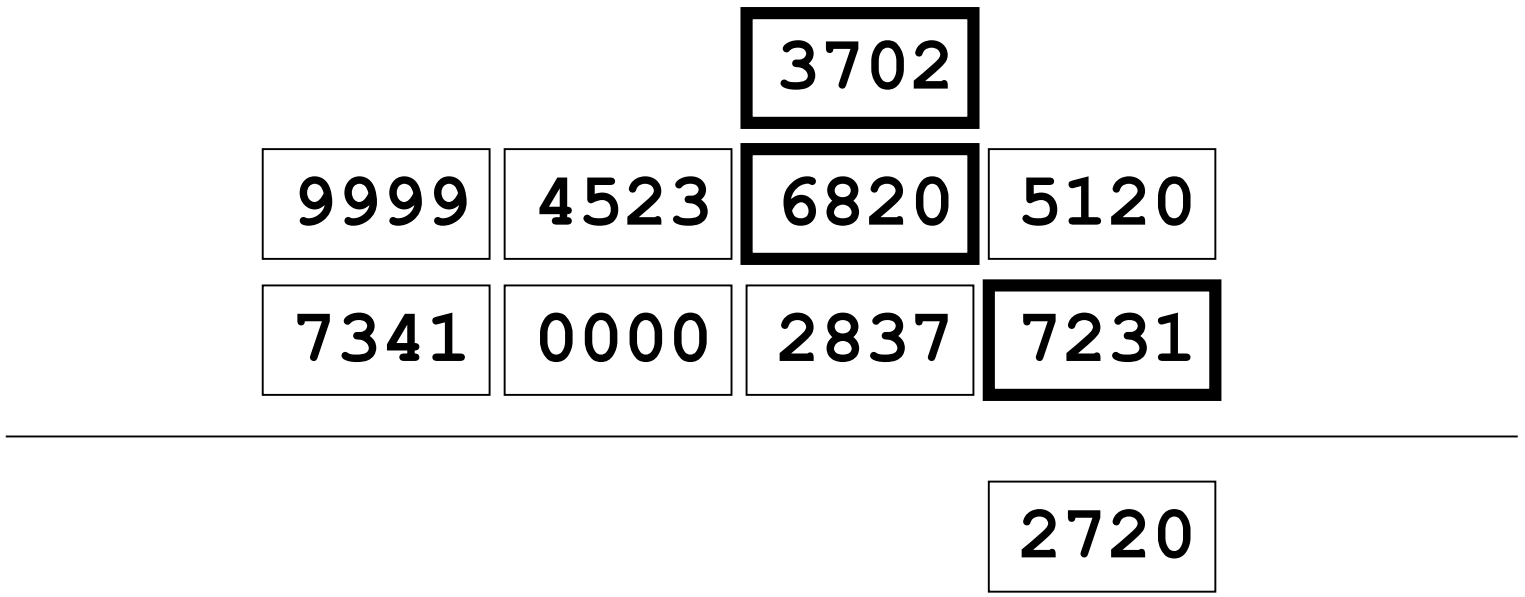
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



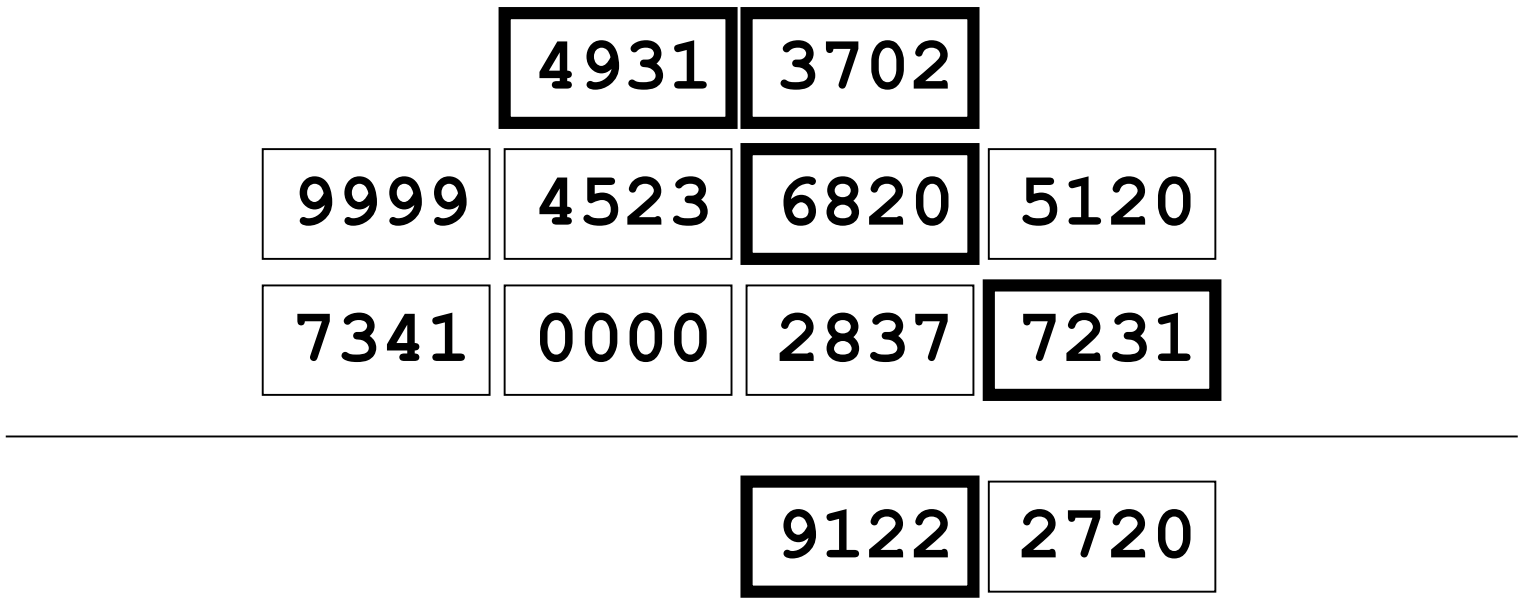
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



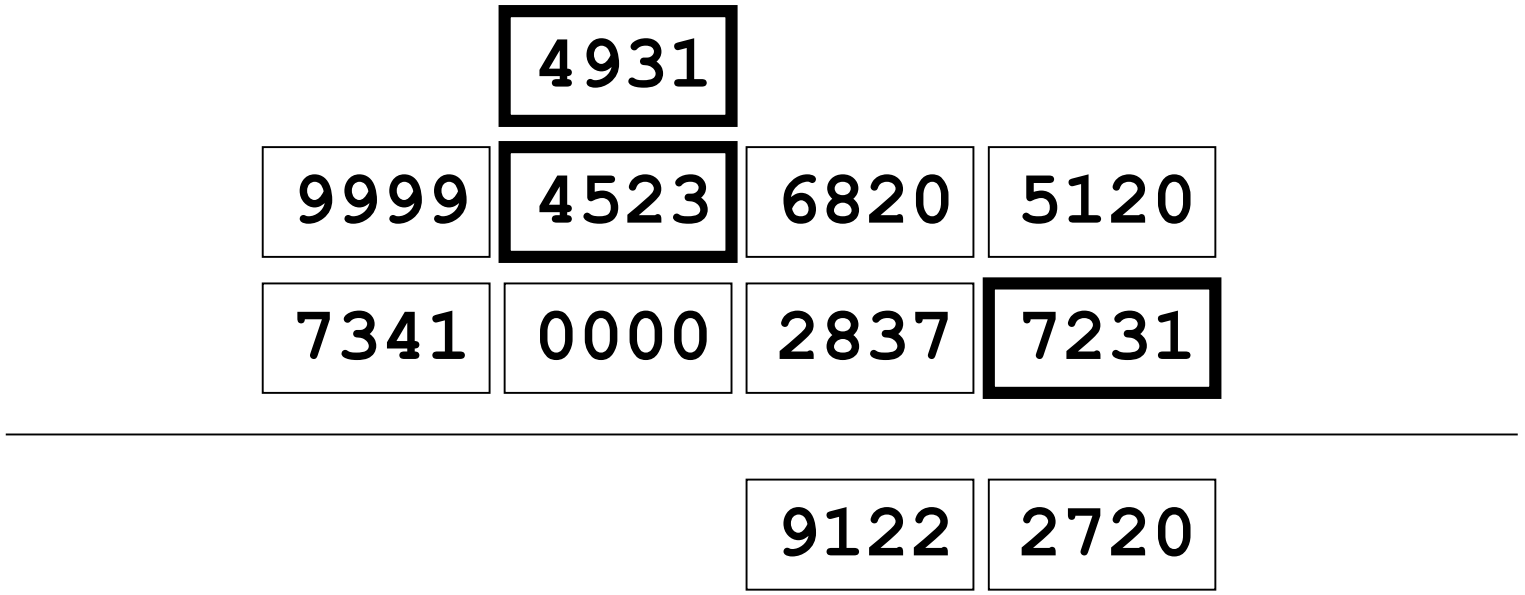
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



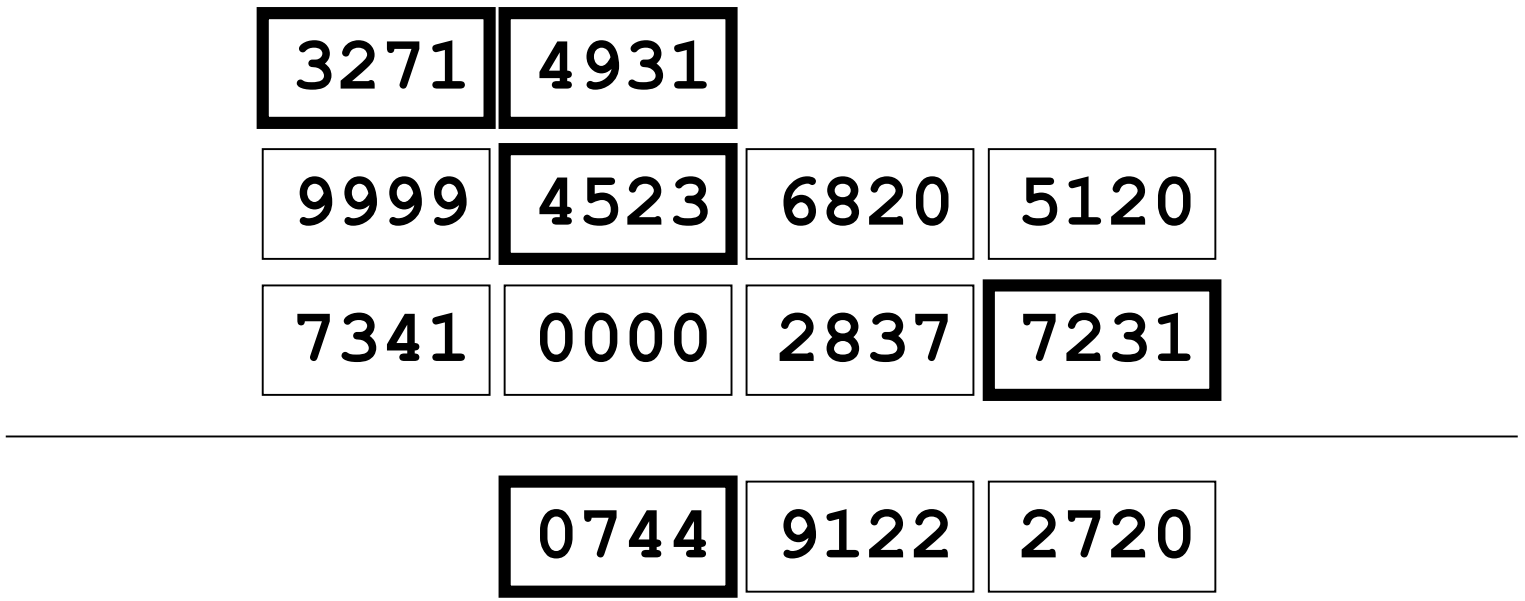
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



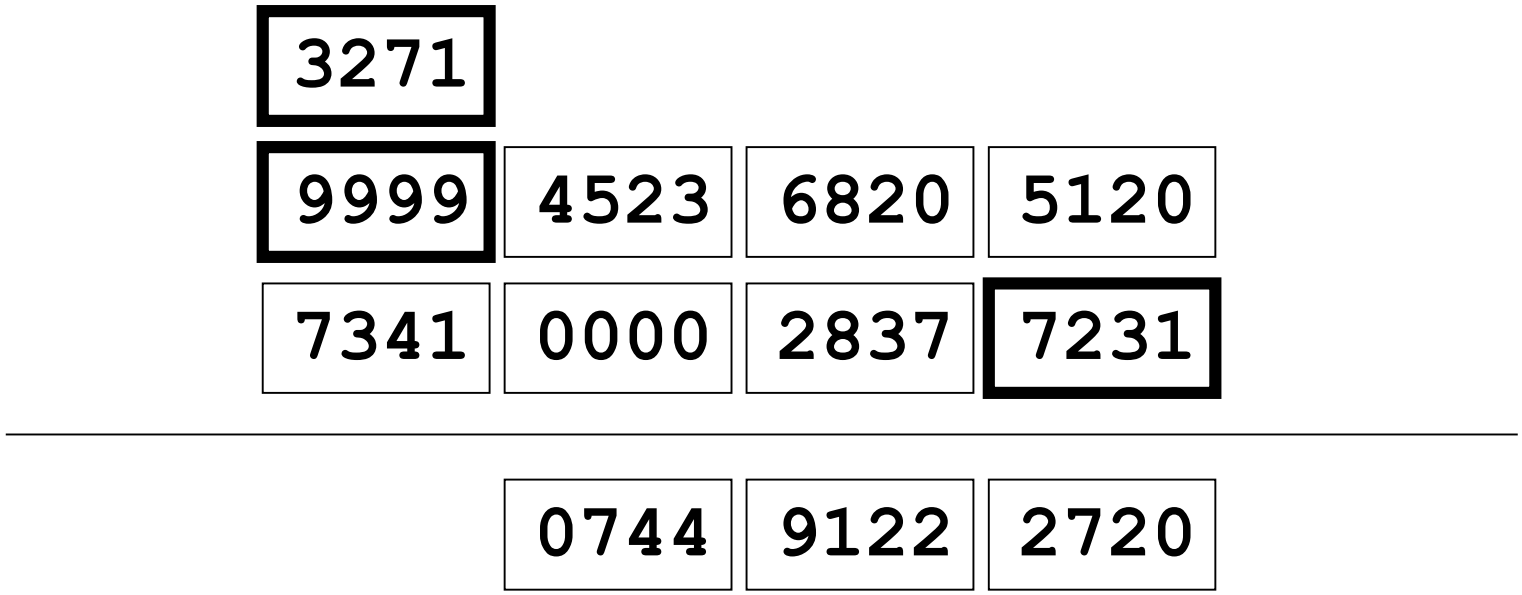
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

7230	**3271**			
	9999	**4523**	**6820**	**5120**
	7341	**0000**	**2837**	**7231**
	6040	**0744**	**9122**	**2720**

Valor acumulado 0

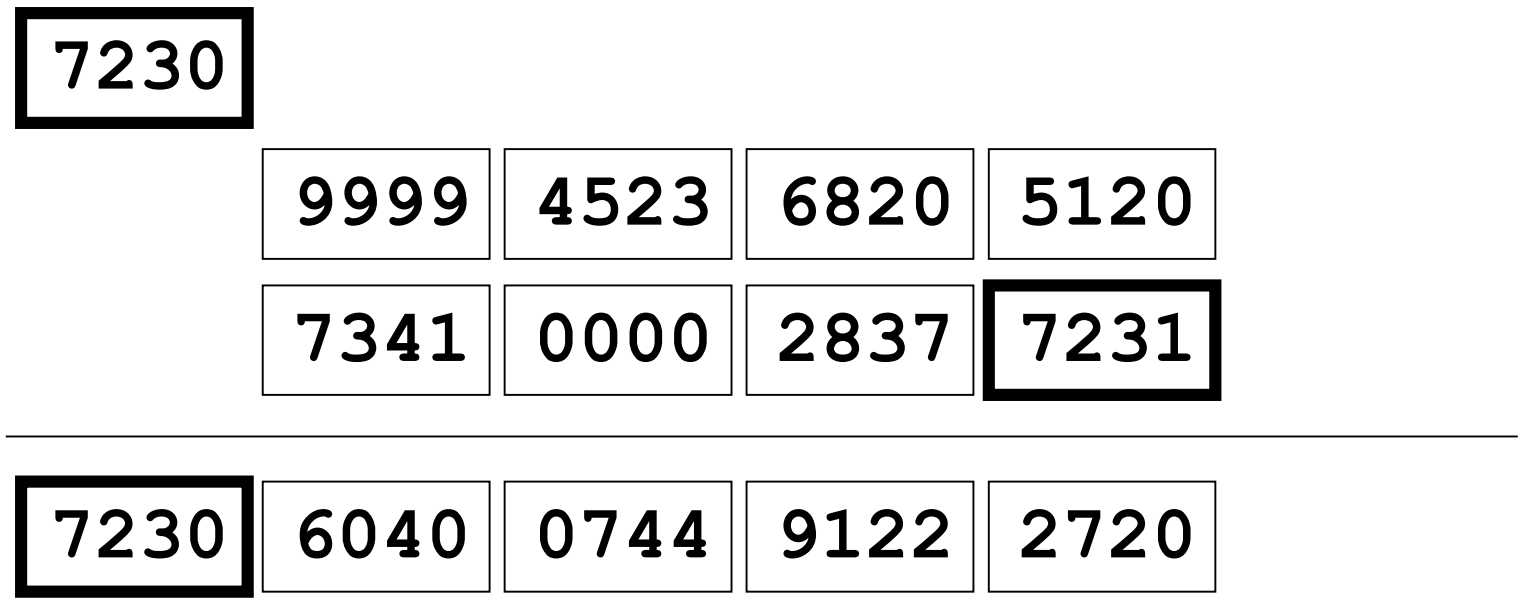
Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

7230			
9999	4523	6820	5120
7341	0000	2837	**7231**

6040	0744	9122	2720

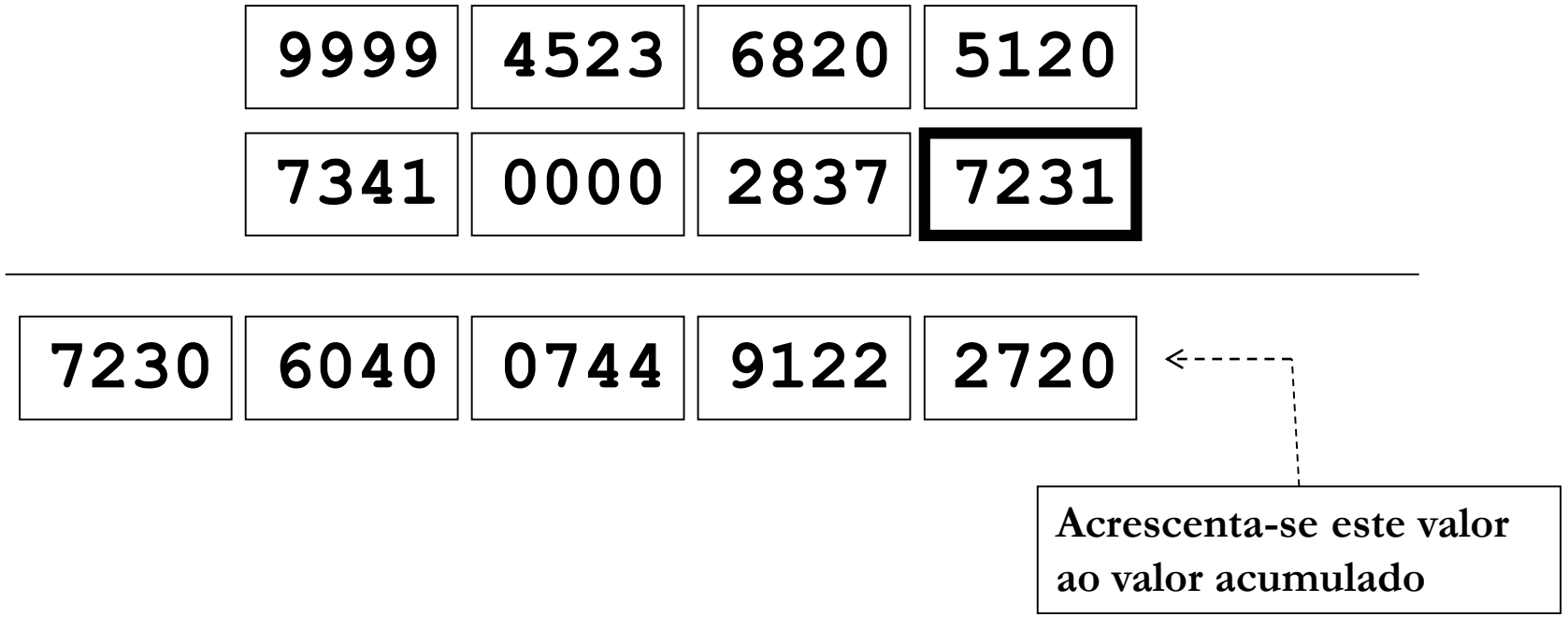
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



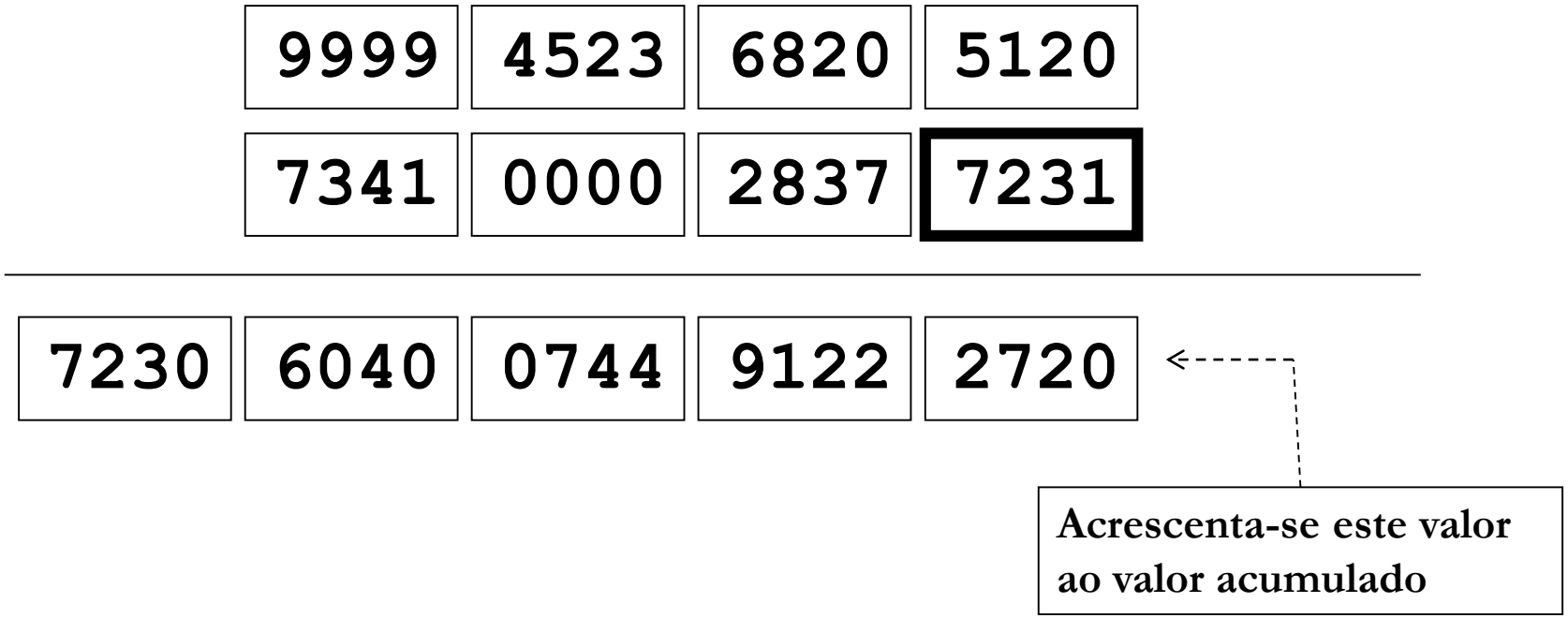
Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 0

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	**5120**
7341	0000	**2837**	7231

Valor acumulado 72306040074491222720

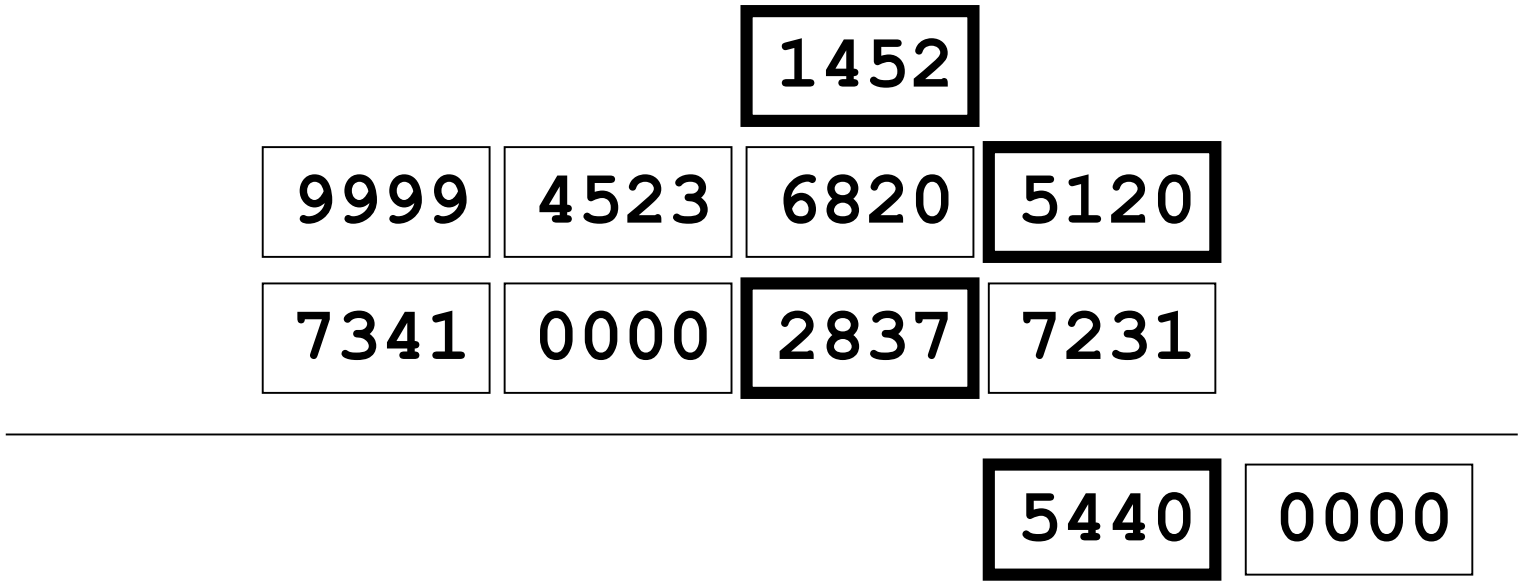
Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	**5120**
7341	0000	**2837**	7231

0000

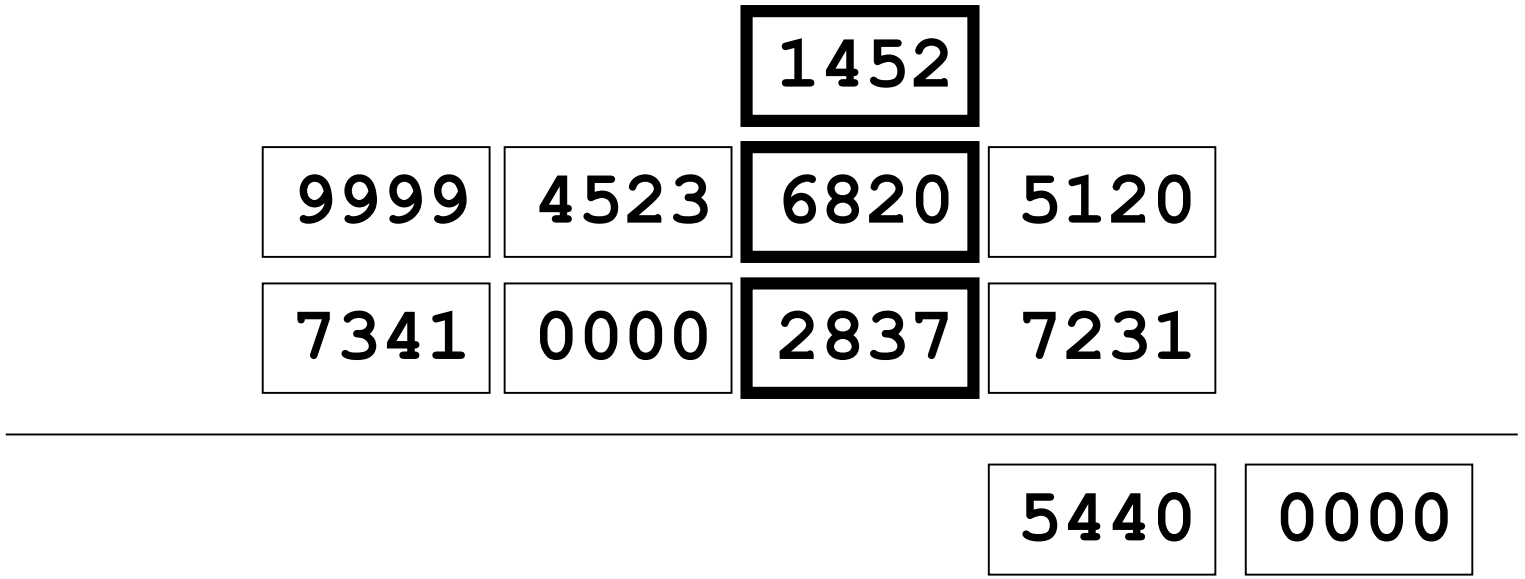
Valor acumulado	72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



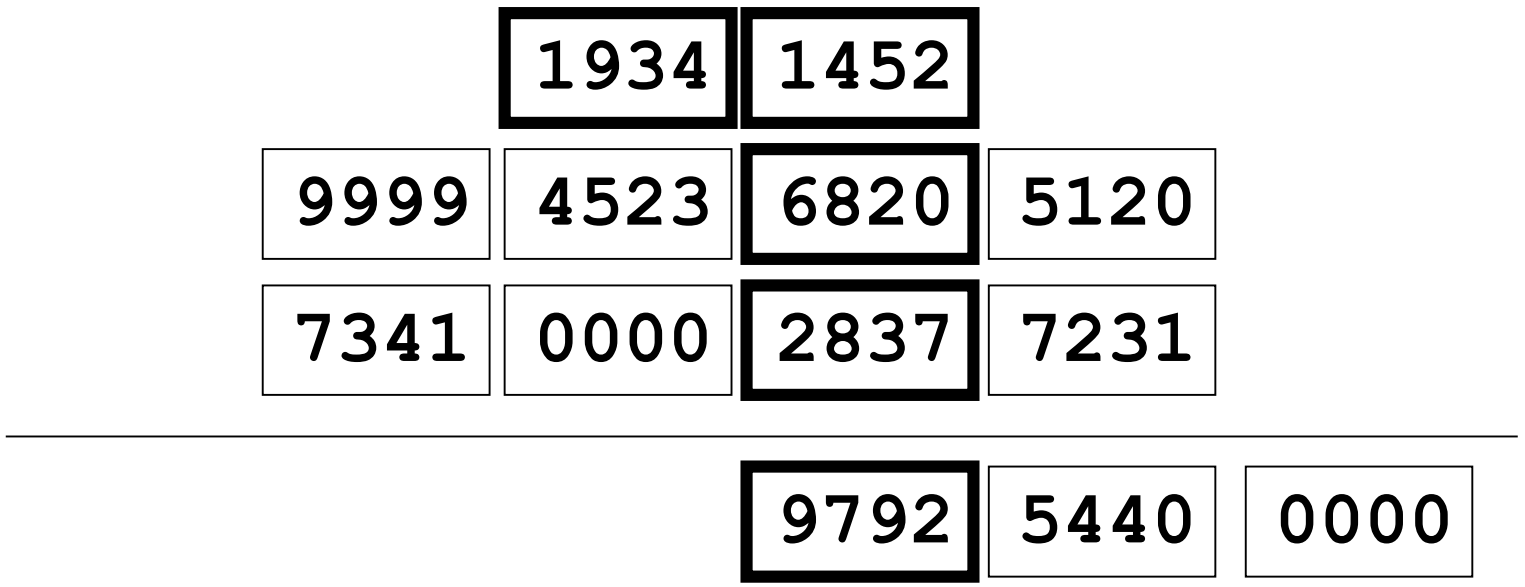
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



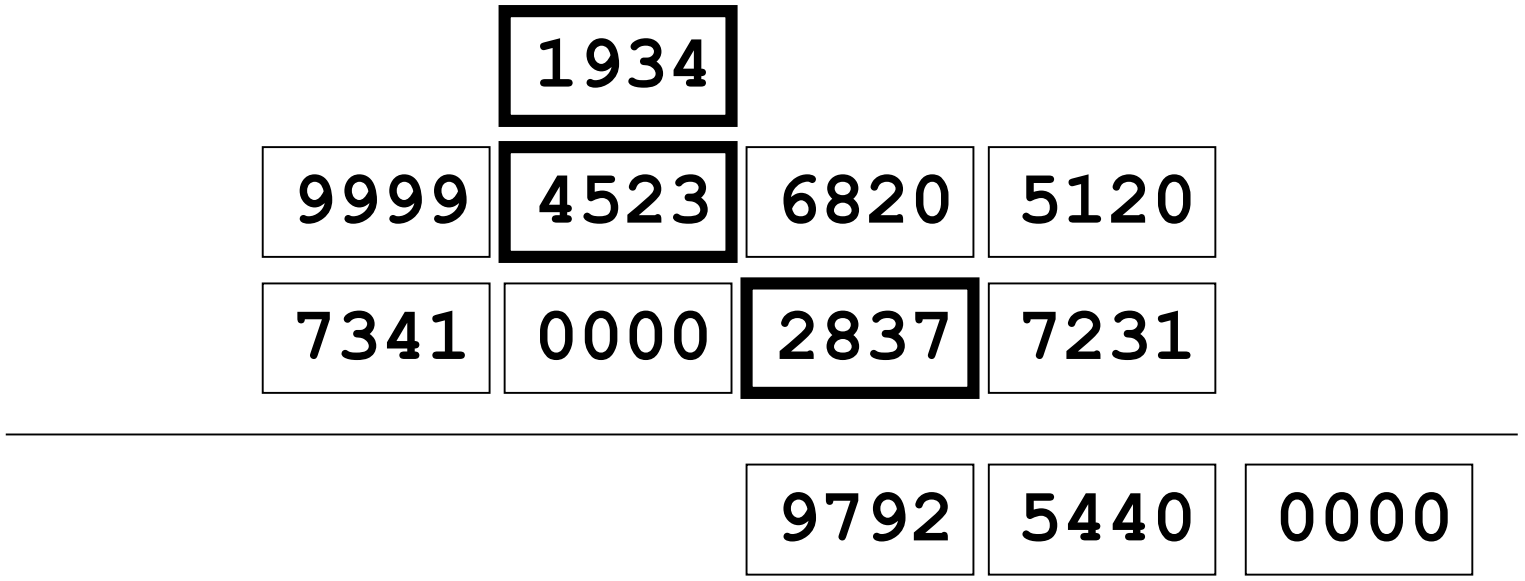
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



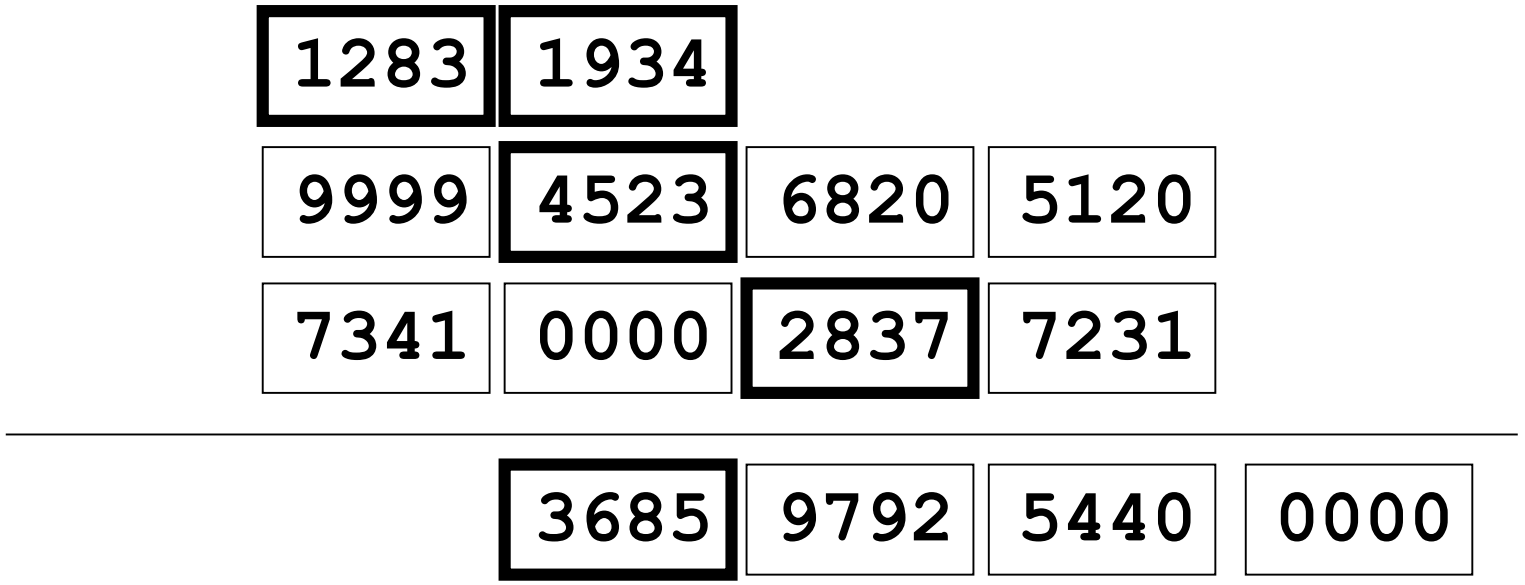
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



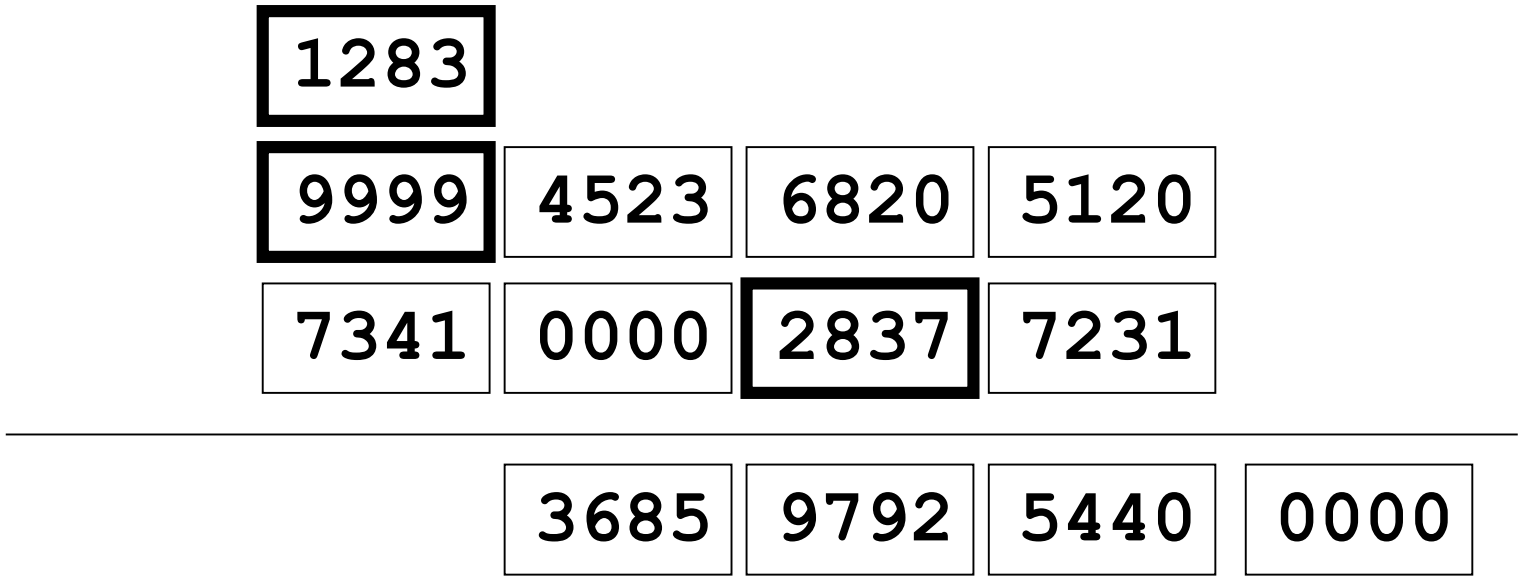
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



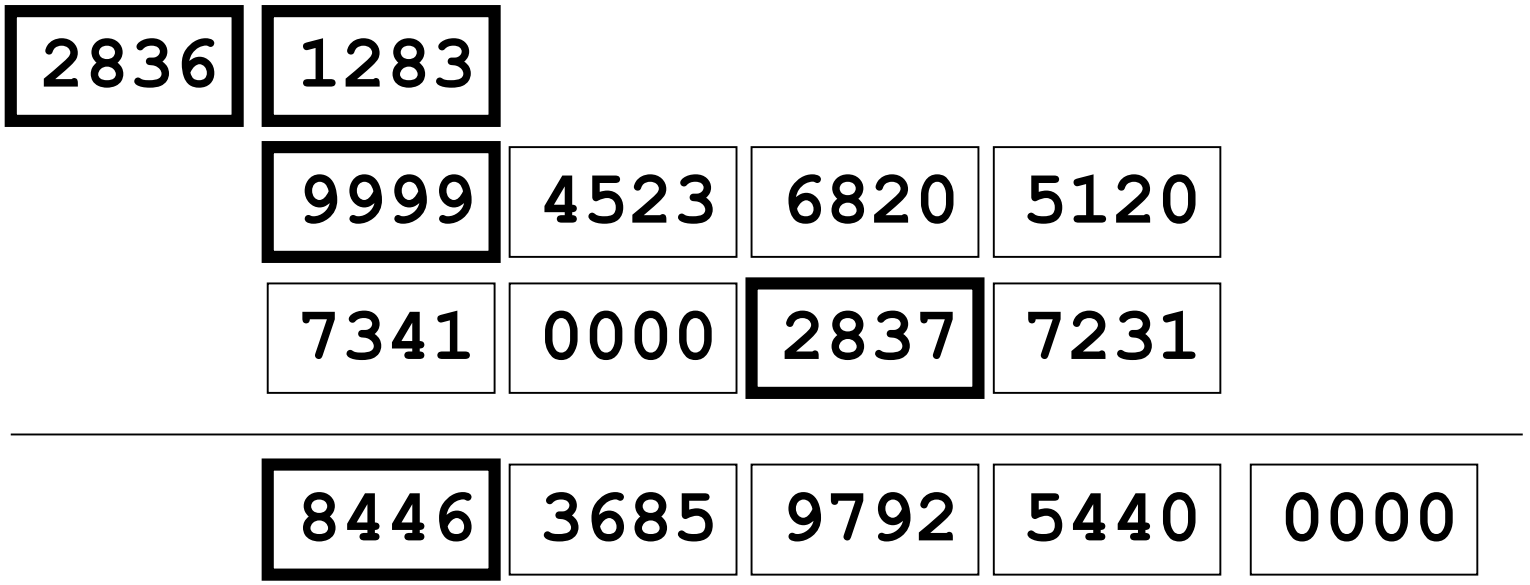
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

2836

9999	4523	6820	5120
7341	0000	**2837**	7231

8446	3685	9792	5440	0000

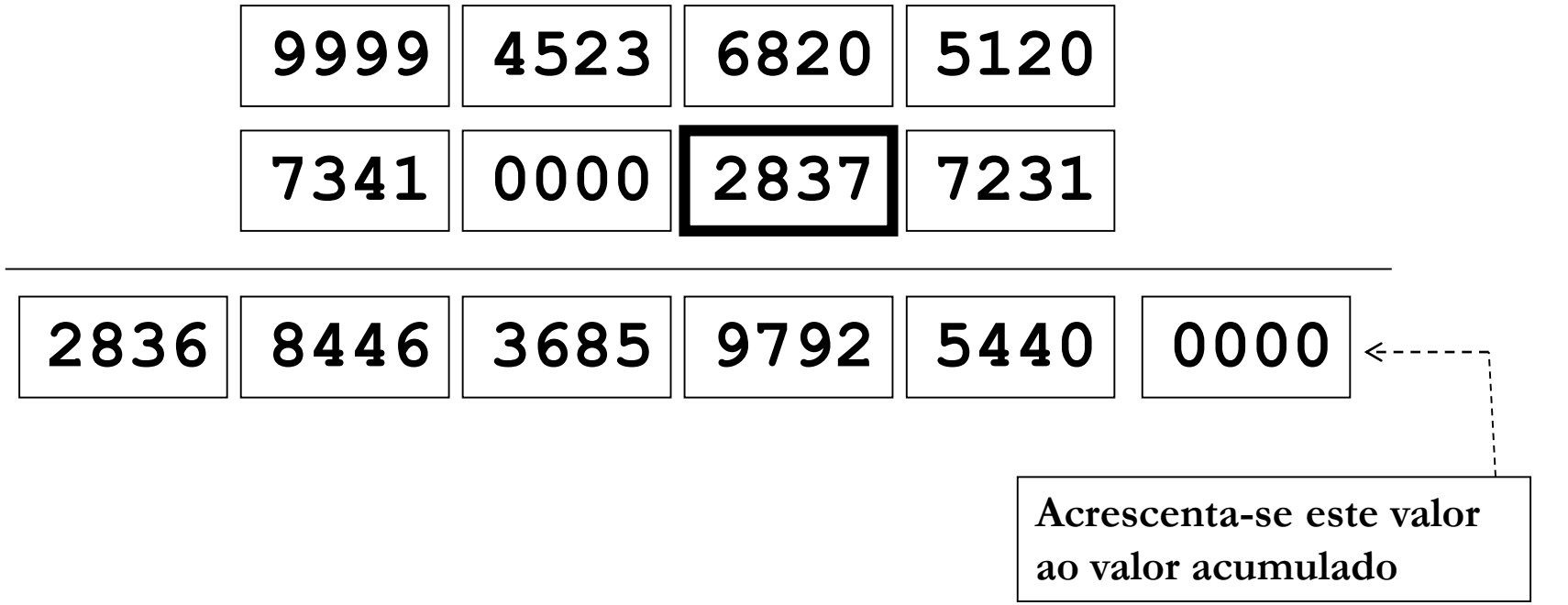
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

2836					
	9999	4523	6820	5120	
	7341	0000	**2837**	7231	
2836	8446	3685	9792	5440	0000

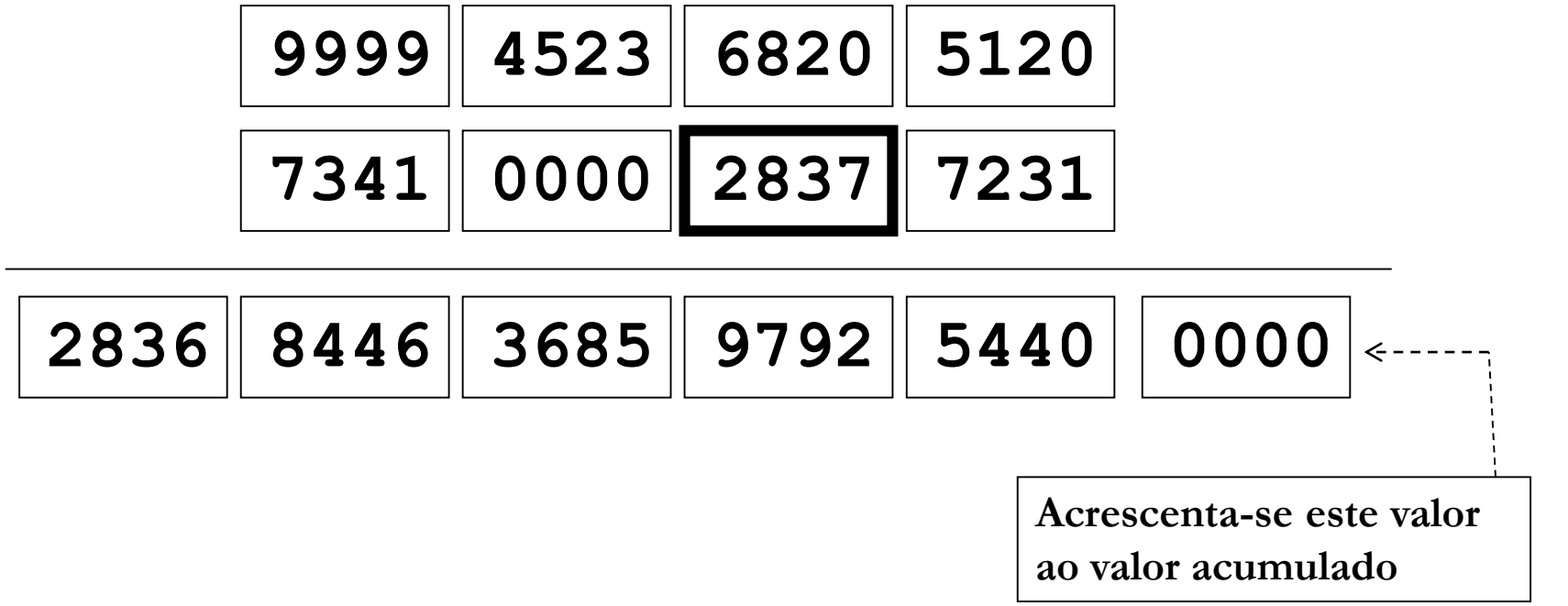
Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 72306040074491222720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



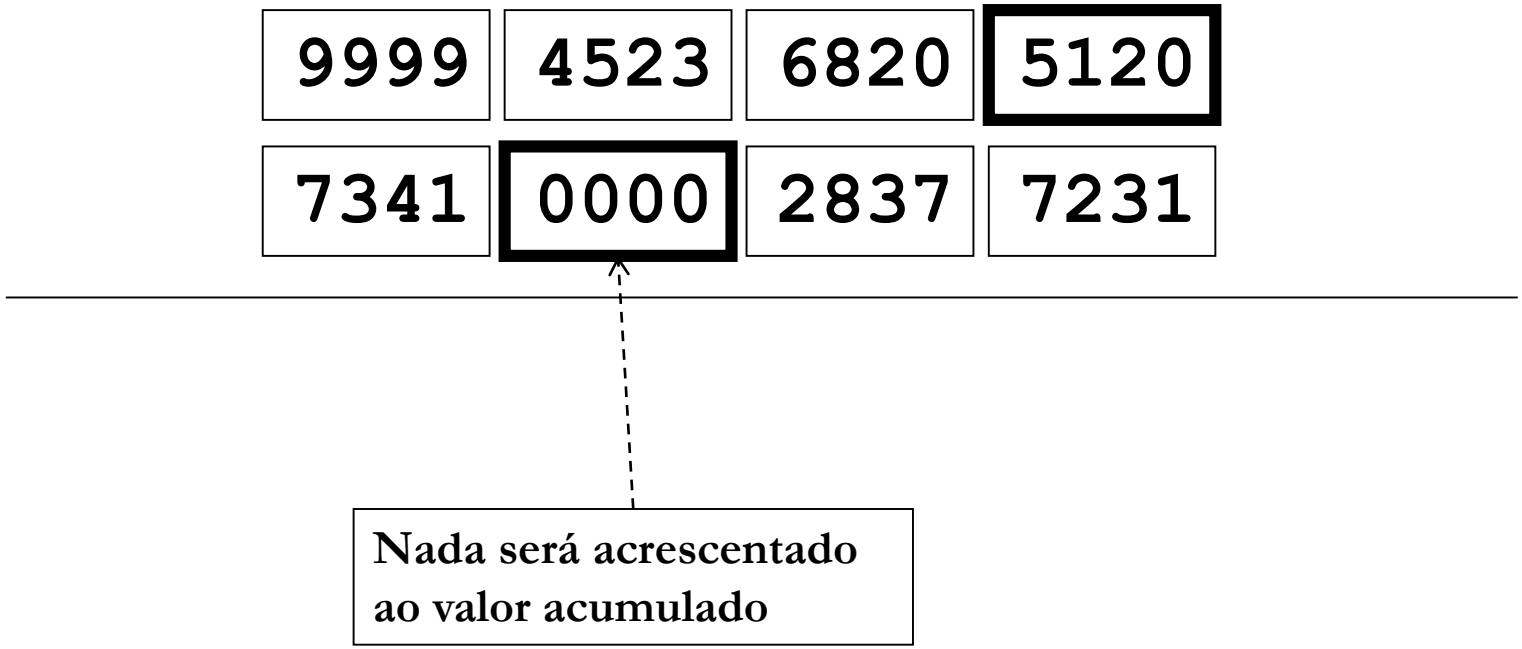
Valor acumulado 28375676972605374562272O

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	**5120**
7341	**0000**	2837	7231

Valor acumulado 283756769726053745622720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**



Valor acumulado 283756769726053745622720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	**5120**
7341	0000	2837	7231

Valor acumulado 283756769726053745622720

Exemplo: Seja a multiplicação dos números **9999452368205120** e **7341000028377231**

9999	4523	6820	**5120**
7341	0000	2837	7231

0000	0000	0000

Valor acumulado 28375676972605374562272O

E assim por diante

A função main:

- Deve **invocar** devidamente as **funções** descritas nos **5 detalhes** anteriores

Observação:

- A **digitação** de cada número de entrada deve ser feita em **linha separada**, ou seja, **cada número** digitado deve ser encerrado com ***<enter>***

0.8 – Bibliografia

0.5.1 – Livros

Livro 01:

Mokarzel, F.C.; Soma, N.Y.

Fundamentos de Programação para Engenharia. São José dos Campos: Instituto Tecnológico de Aeronáutica, 2020.



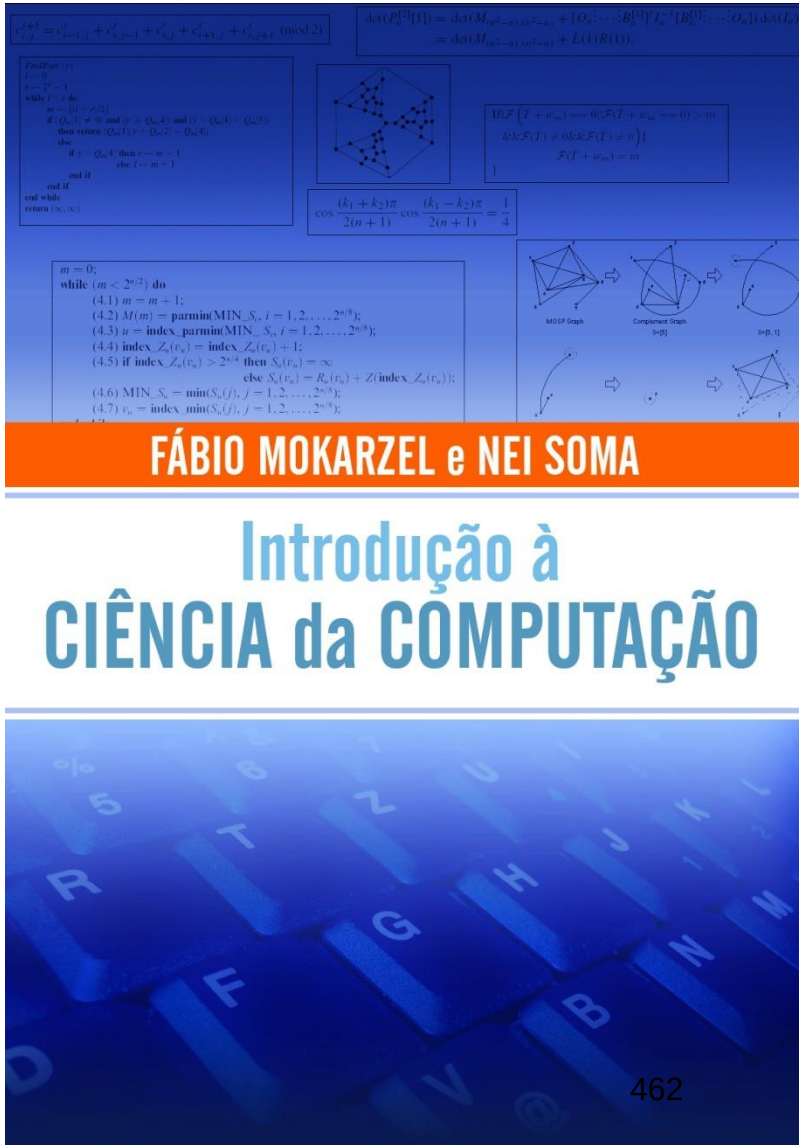
0.5 – Bibliografia

0.5.1 – Livros

Livro 02:

Mokarzel, F.C.; Soma, N.Y.

Introdução à Ciência da Computação. Rio de Janeiro: Campus-Elsevier, 2008.



0.5 – Bibliografia

0.5.1 – Livros

Livro 03:

Mizrahi, V.V. ***Treinamento em Linguagem C***. 2ª Edição. São Paulo: Pearson, 2008.

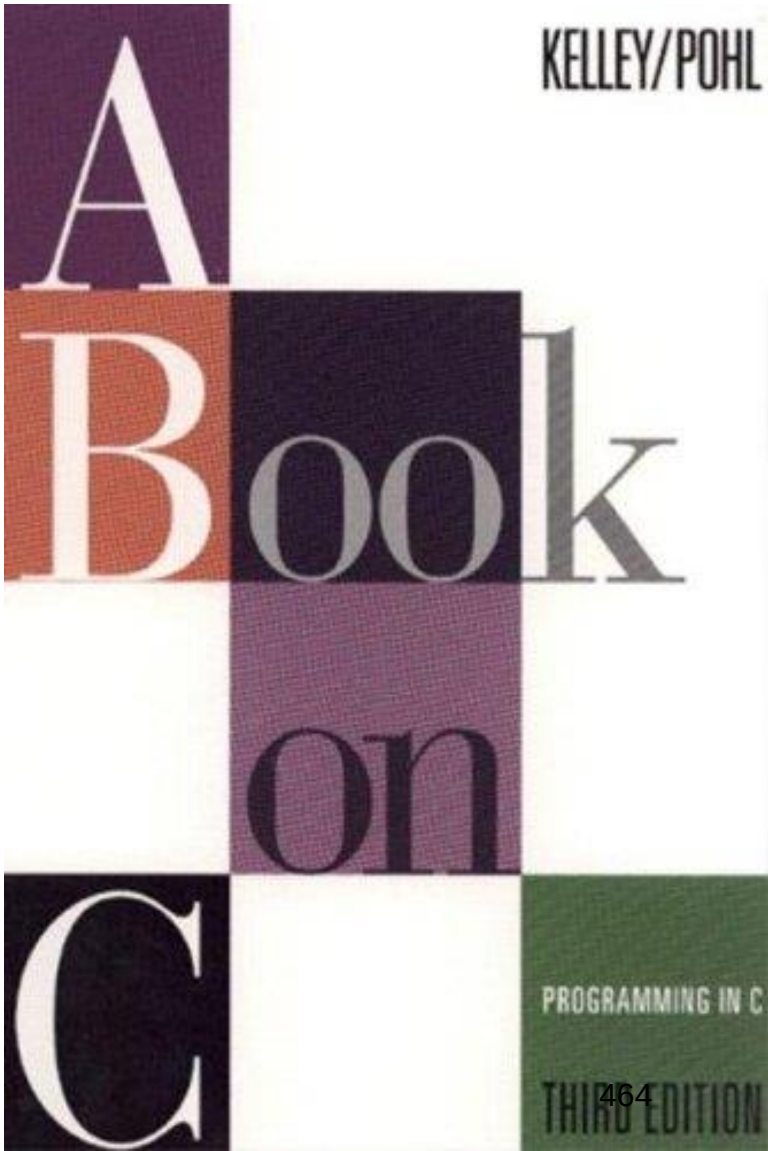


0.5 – Bibliografia

0.5.1 – Livros

Livro 04:

Kelley, A.; Pohl, I. ***A Book on C***. Redwood City: Benjamin Cummings, 1995.



0.5 – Bibliografia

0.5.1 – Livros

Livro 05:

Saliba, W.L.C. ***Técnicas de Programação: uma Abordagem Estruturada***. São Paulo: Makron, 1992.

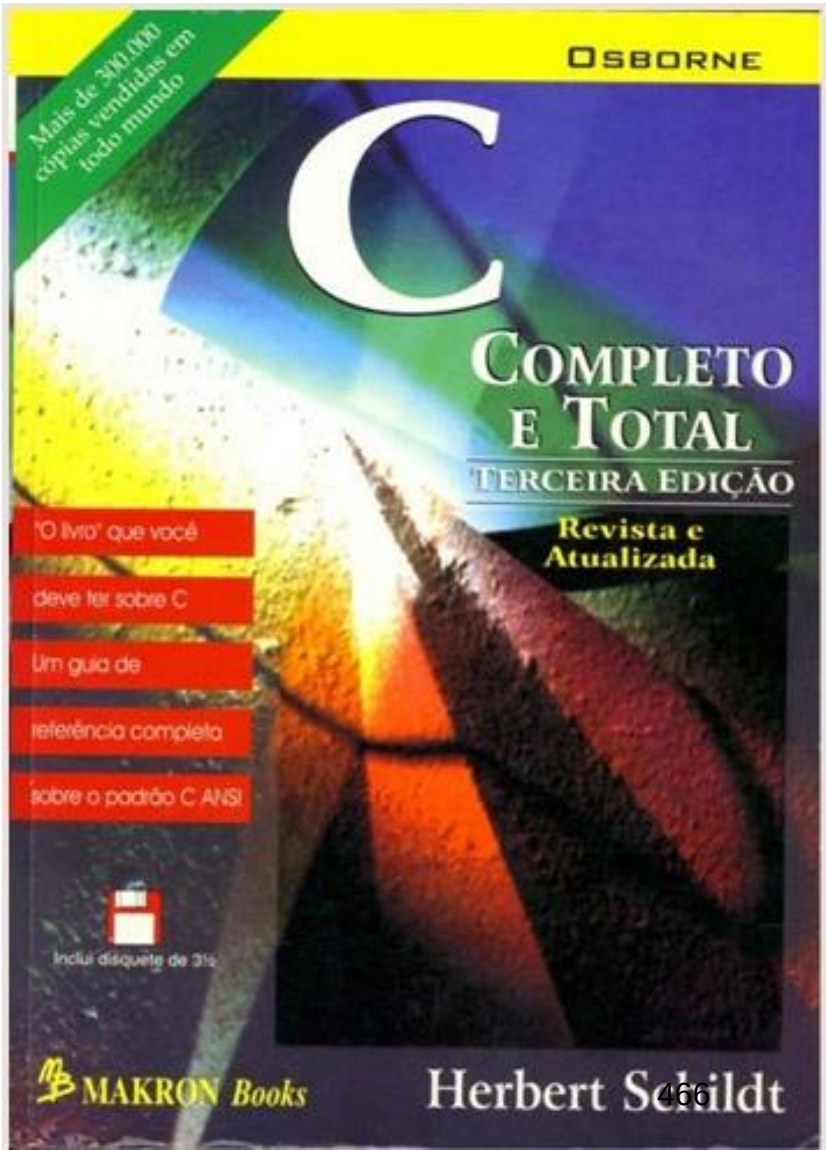


0.5 – Bibliografia

0.5.1 – Livros

Livro 06:

Schildt, H. ***C Completo e Total***. 3ª Edição São Paulo: Makron, 1997.

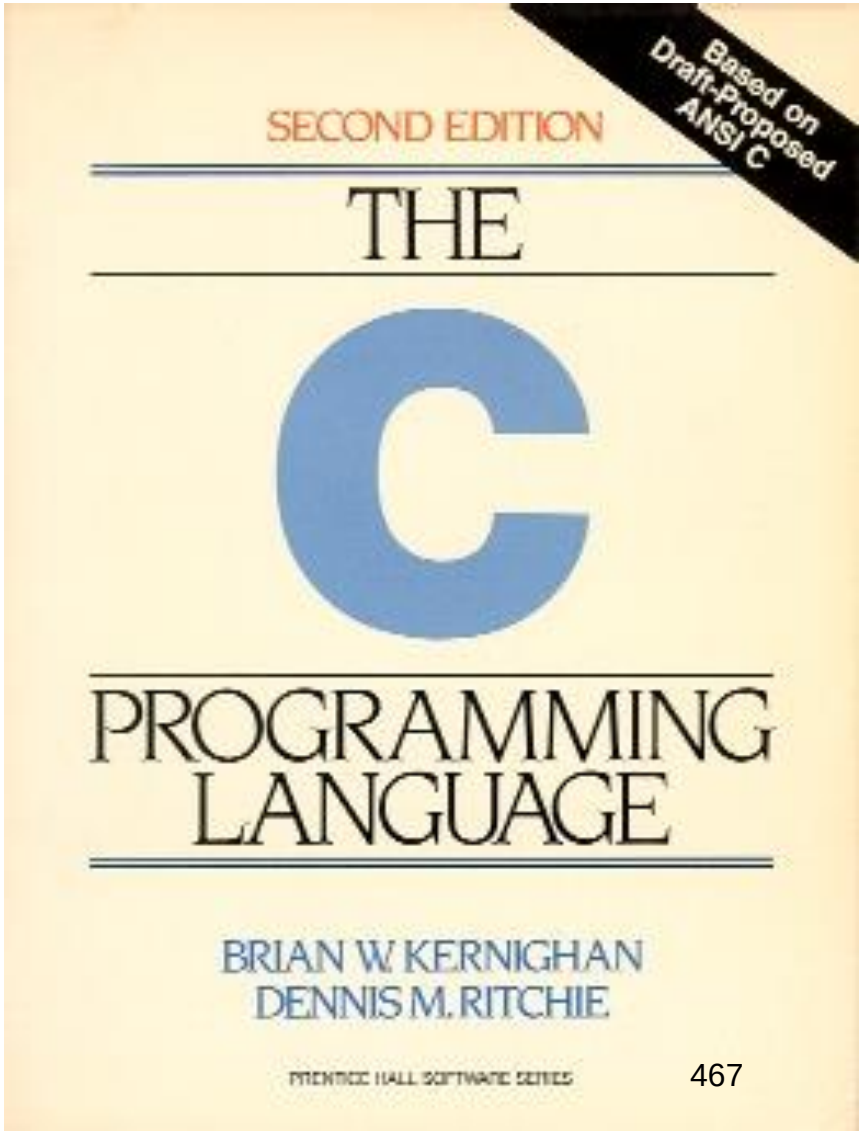


0.5 – Bibliografia

0.5.1 – Livros

Livro 07:

Kernighan B.W.; Ritchie D.M. ***The C Programming Language***. 2nd Edition Englewood Cliffs: Prentice Hall, 1988.

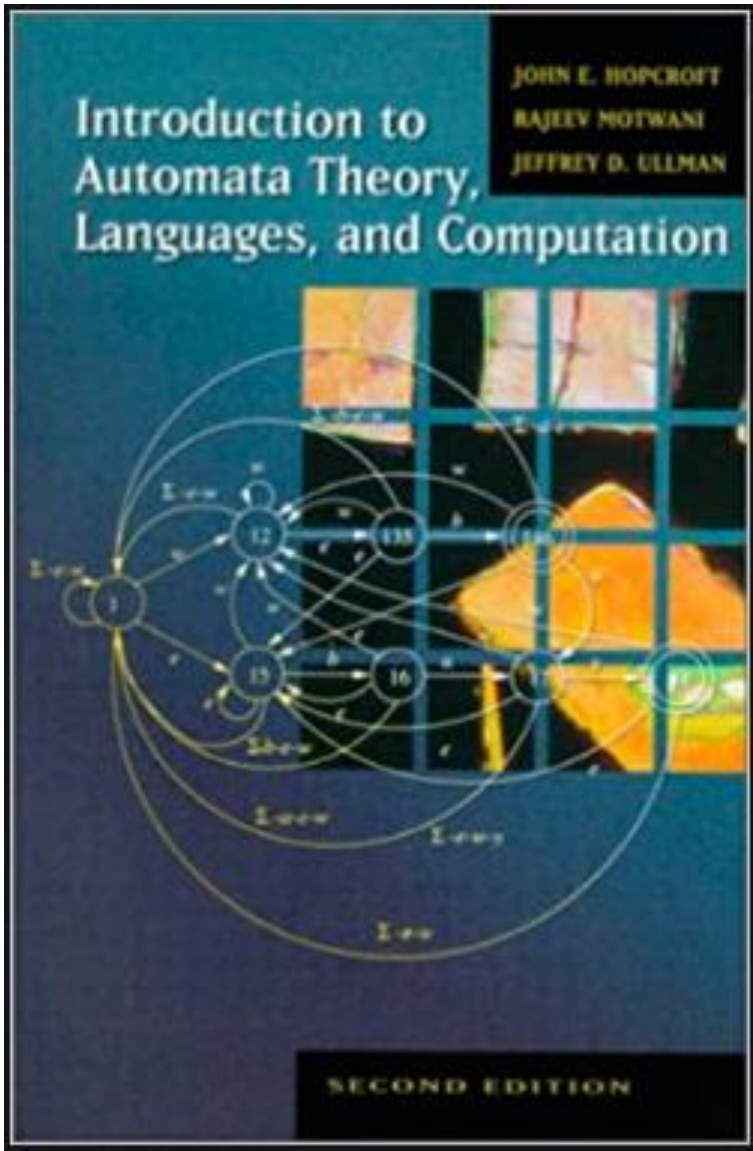


0.5 – Bibliografia

0.5.1 – Livros

Livro 08:

Hopcroft, J.E.; Motwani, R.J.; Ullman, D. ***Introduction to Automata Theory, Languages and Computations***, 2nd Edition, Addison Wesley, 2001.



0.5 – Bibliografia

0.5.2 – Slides em Power-Point

PPT 01:

Slides de CES-10 –

Prof. Mokarzel, 2011



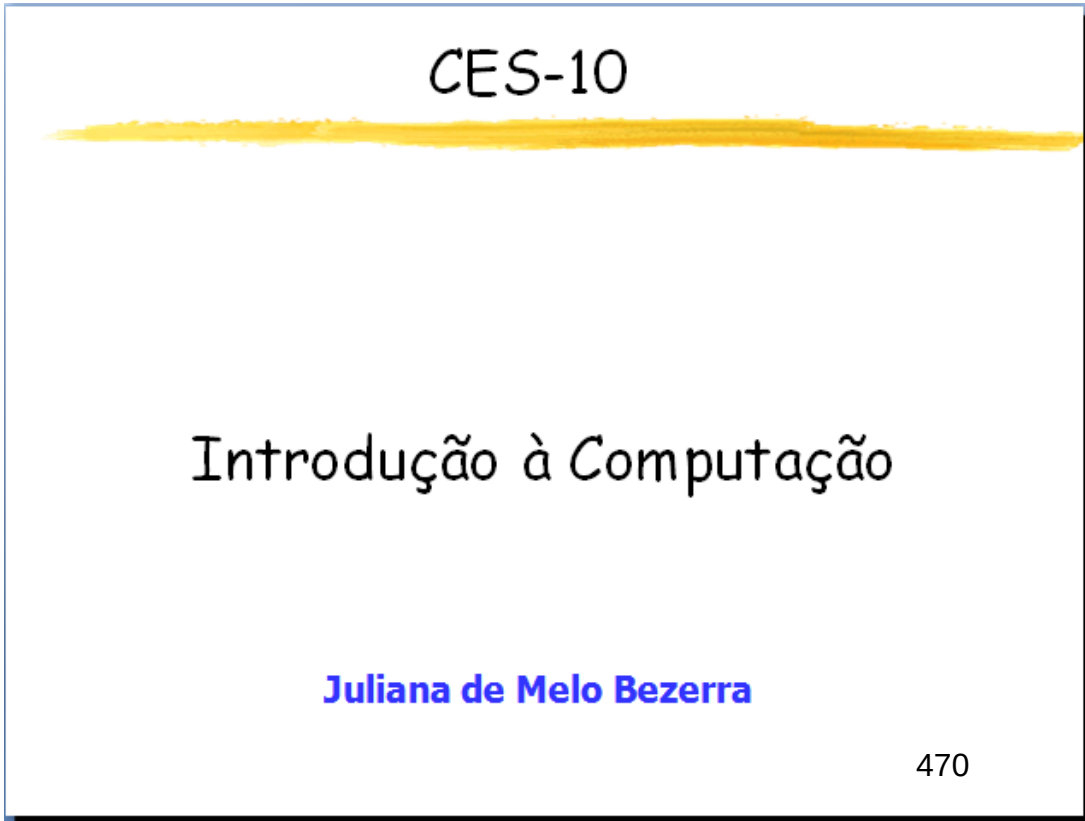
0.5 – Bibliografia

0.5.2 – Slides em Power-Point

PPT 02:

Slides de CES-10 –

Profa. Juliana, 2009



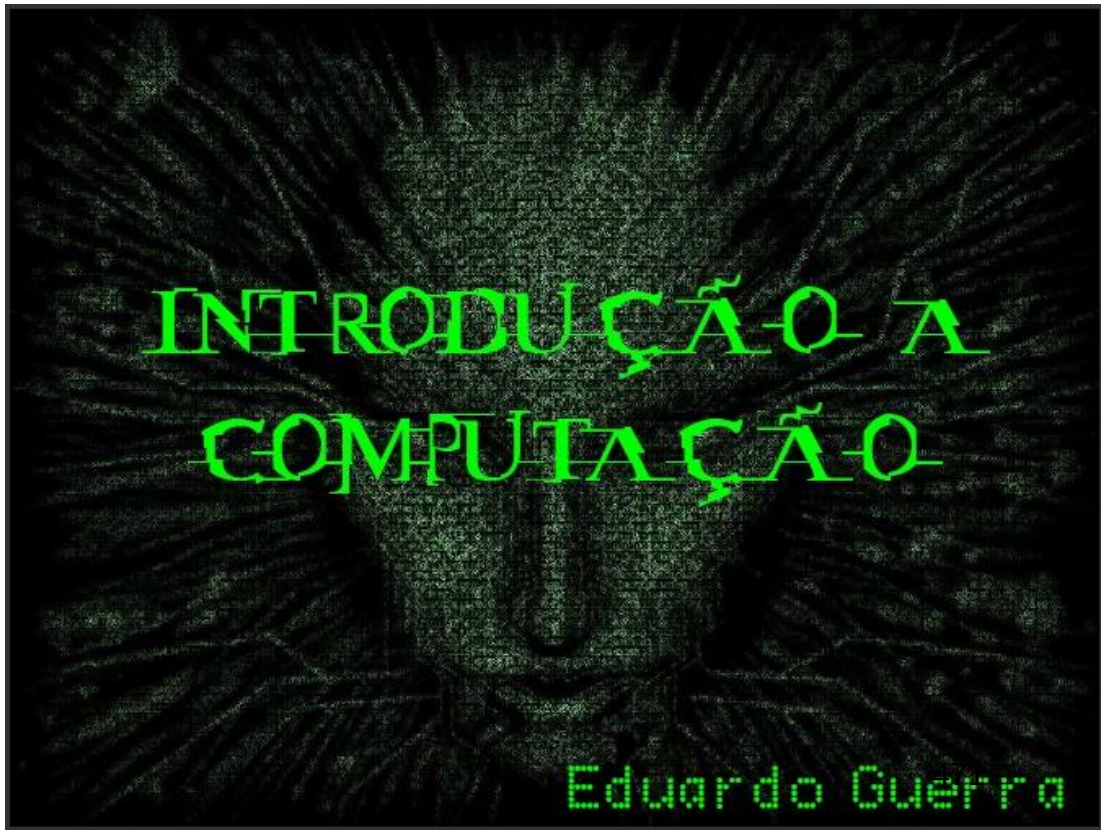
0.5 – Bibliografia

0.5.2 – Slides em Power-Point

PPT 03:

Slides de CES-10 –

Prof. Guerra, 2009



Fábio Carneiro Mokarzel

Professor na Divisão de Ciência da Computação do Instituto Tecnológico de Aeronáutica (ITA) desde 1978. Chefe do Departamento de Engenharia de Software do ITA de 1996 a 2015. Coordenador do Curso de Engenharia de Computação do ITA de 1999 a 2011. Engenheiro Elétrico pela UNIFEI, mestre e doutor em Computação pelo ITA.

Nei Yoshihiro Soma

Professor da Divisão de Ciência da Computação do Instituto Tecnológico de Aeronáutica (ITA) desde 1986. Chefe da mesma divisão de 2003 a 2017. Mestre em Análise de Sistemas pelo INPE, Doutor em Matemática Aplicada e Computacional pela Universidade de Sheffield, pesquisador visitante da Universidade de Bolonha. Participou do Comitê Assessor do CNPq, coordenou as Engenharias III e foi titular do Conselho Técnico Científico da Educação Superior na Capes.

Aulas Práticas de Introdução à Programação

Atualmente, disciplinas introdutórias de Computação e Programação têm importância muito grande em cursos de Engenharia de praticamente todas as modalidades, devido ao fato de os sistemas computacionais estarem integrados em todas as atividades científicas e tecnológicas da sociedade. No entanto, o ensino dessas disciplinas em diversas escolas de Engenharia tem sido delegado a novos professores que, em muitos casos, têm dificuldades em preparar material teórico e prático sobre o assunto, que seja abrangente, substancial e agradável de ser acompanhado.

Previamente foi lançado pelos autores um conjunto de três livros que destina-se a auxiliar esses professores, tendo sido fruto de longa experiência dos autores na ministração dessas disciplinas. Já este livro vem complementar o referido conjunto com a proposição de diversas atividades práticas, visando proporcionar aos alunos considerável experiência de programação unimodular e multimodular. Essas atividades variam desde a elaboração de programas bem simples, como a construção de tabelas de funções de uma variável, passando por programas contendo aninhamentos de comandos condicionais e repetitivos, programas envolvendo variáveis escalares, estruturadas homogêneas e heterogêneas, e programas contendo subprogramas, recursividade e encadeamento de estruturas por ponteiros.