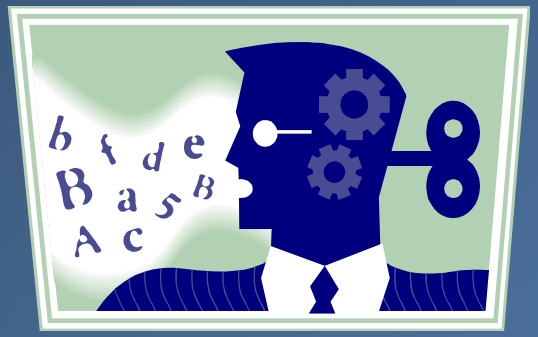


Conceitos Elementares

Revisão: estruturas de controle

Estruturas Básicas de Controle

- Segundo o Paradigma da Programação Estruturada, qualquer programa pode ser descrito através de três estruturas básicas de controle:

  - **Instruções de Sequência**: instruções atômicas (simples) que permitem leitura/escrita de dados, cálculos e atribuições de valores

  - **Instruções de Decisão:** permitem a seleção de um ou outro conjunto de ações, após a avaliação lógica de uma condição

  - **Instruções de Repetição:** permitem a execução repetitiva de um bloco de instruções. O número de repetições é controlado pelo valor de uma condição lógica, que é testado a cada iteração

Estrutura Sequencial

- Ordenação linear dos comandos em um algoritmo, significando que deve ser executados seguindo-se o texto em que estão escritos, de cima para baixo.

Algoritmo
d_1
...
d_n
c_1
...
c_n
fim algoritmo

Ex:

```
Algoritmo
    declare A, B, C numérico
    leia A, B
    C ← (A+B) × B
    escreva A, B, C
fim algoritmo
```

Estrutura Sequencial

- **Exercício**:
- Passar para a linguagem C

```
Algoritmo
        declare A, B, C numérico
        leia A, B
        C ← (A+B) × B
        escreva A, B, C
fim algoritmo
```

Estrutura Condicional

- Permite a escolha do grupo de ações a ser executado
- Escolha baseada na satisfação ou não de determinadas condições
- Condições são representadas por expressões lógicas
- Dois tipos de estruturas condicionais:
  - Estrutura Condicional Simples
  - Estrutura Condicional Composta

Estrutura Condicional Simples

- Determinada sequência de comandos só será executada se a condição for verdadeira

- Forma Geral:

```
se condição
  então sequência de comandos
fim se
```

- Exemplo:

```
se (MEDIA ≥ 6.0)
  então escreva "Aprovado!"
fim se
```

Em C:

```
if (MEDIA >= 6.0) {
    printf("Aprovado");
}
```

Estrutura Condicional Composta

- Sequência A de comandos é executada se a condição for verdadeira, e sequência B se a condição for falsa

- Forma Geral:

```
se condição
        então sequência A de comandos
        senão sequência B de comandos
fim se
```

- Exemplo:

```
se (MEDIA ≥ 6.0)
  então escreva "Aprovado!"
  senão escreva "Reprovado!"
fim se
```

Em C:

```
if (MEDIA >= 6.0) {
    printf("Aprovado");
}
else {
    printf("Reprovado");
}
```

Fixação

- Fazer algoritmo que leia 2 números (N_1 e N_2) e, se ambos forem negativos, escrever a soma deles, senão escrever o produto.

- Fazer algoritmo que leia 3 valores numéricos, determine e imprima o menor deles.

SUGESTÃO:

- Implementar os algoritmos em C

Estrutura de Repetição

- Permite a repetição de um conjunto de comandos
- Os comandos são repetidos até que uma condição de término seja satisfeita
- Condições são representadas por expressões lógicas
- Dois tipos de estruturas de repetição:
    - Repetição com teste da condição no início
    - Repetição com teste da condição no fim

Repetição com teste no início

- O teste é feito antes que qualquer comando do bloco seja executado.

- Forma Geral:

```
enquanto condição verdadeira faça
  sequência de comandos
  ...
fim enquanto
```

- Ex: calcular a soma dos nos pares de 100 a 200, inclusive:

```
declare SOMA, PAR numérico
PAR  ← 100
SOMA ← 0
enquanto PAR ≤ 200 faça
  SOMA ← SOMA + PAR
  PAR  ← PAR + 2
fim enquanto
```

Em C:

```
int SOMA, PAR;
PAR = 100;
SOMA = 0;
while (PAR <= 200) {
    SOMA = SOMA + PAR;
    PAR = PAR + 2;
}
```

Repetição com teste no final

- O teste é feito depois que os comandos do bloco são executados (ao menos uma vez).

- Forma Geral:

```
repita
  sequência de comandos
  ...
enquanto condição verdadeira
```

- Ex: calcular a soma dos nos pares de 100 a 200, inclusive:

```
declare SOMA, PAR numérico
PAR  ← 100
SOMA ← 0
repita
  SOMA ← SOMA + PAR
  PAR  ← PAR + 2
enquanto PAR ≤ 200
```

Em C:

```
int SOMA, PAR;
PAR = 100;
SOMA = 0;
do {
     SOMA = SOMA + PAR;
     PAR = PAR + 2;
} while (PAR <= 200);
```

Fixação

- Que valores são escritos pelo algoritmo abaixo:

```
Algoritmo
    declare N, QUADRADO  numérico
    N ← 10
    repita
      QUADRADO ← N²
      escreva QUADRADO
      N ← N - 1
    até que N < 1
fim algoritmo
```

- Fazer um algoritmo que leia um número indeterminado de idades, de um grupo de indivíduos, calcule e escreva a idade média deste grupo.
  - Quanto uma idade igual a 0, ou negativa, for informada, ela não entra nos cálculos e o algoritmo finaliza

Constantes, Variáveis, Tipos de Dados

Tipos de Dados

- *tipo inteiro* caracteriza qualquer dado numérico que pertença ao conjunto dos números inteiros
- *tipo real* caracteriza qualquer dado numérico que pertença ao conjunto dos números reais
- *tipo caracter* caracteriza qualquer dado que pertença a um conjunto de caracteres *alfanuméricos*
- *tipo lógico* caracteriza qualquer dado que possa assumir somente uma de duas situações: verdadeiro ou falso

Tipos de Dados em C

- `int a; /* números inteiros com no máximo 4 bytes, incluindo o sinal [-2147483648 a 2147483647]. */`
- `unsigned int b; /* números inteiros sem sinal com no máximo 4 bytes. [0 a 4294967295]. */`
- `short c; /* inteiros com no máximo 2 bytes, incluindo o sinal [-32768 a 32767]. */`
- `unsigned short d; /* inteiros sem sinal com no máximo 2 bytes [0 a 65535]. */`
- `float e; /*números reais com no máximo 4 bytes, incluindo o sinal*/`
- `double f; /*números reais com no máximo 8 bytes, incluindo o sinal*/`
- `char g; /* caracteres alfanuméricos com no máximo 1 byte, incluindo o sinal no caso de número inteiro. Exs: 'a', 'X', '%' e números de -128 a 127.*/`

- Alguns sistemas operacionais consideram apenas 2 bytes para o tipo int. Neste caso, o tipo *long* estende para 4 bytes.
  - Verificar com `sizeof(tipo)`

Exercício

- *tipo inteiro* caracteriza qualquer dado numérico que pertença ao conjunto dos números inteiros
- *tipo real* caracteriza qualquer dado numérico que pertença ao conjunto dos números reais
- *tipo caracter* caracteriza qualquer dado que pertença a um conjunto de caracteres *alfanuméricos*
- *tipo lógico* caracteriza qualquer dado que possa assumir somente uma de duas situações: verdadeiro ou falso

- Quais os tipos dos dados abaixo?

- Nome do Aluno
- Nota 1
- Nota 2
- Média
- Aprovação

Exercício: solução

- *tipo inteiro* caracteriza qualquer dado numérico que pertença ao conjunto dos números inteiros
- *tipo real* caracteriza qualquer dado numérico que pertença ao conjunto dos números reais
- *tipo caractere* caracteriza qualquer dado que pertença a um conjunto de caracteres *alfanuméricos*
- *tipo lógico* caracteriza qualquer dado que possa assumir somente uma de duas situações: verdadeiro ou falso

- Quais os tipos dos dados abaixo?

		Pseudocódigo:
• Nome do Aluno	*Tipo caractere*	*Alfanumérico*
• Nota 1	*Tipo real*	*Numérico real*
• Nota 2	*Tipo real*	*Numérico real*
• Média	*Tipo real*	*Numérico real*
• Aprovação	*Tipo caractere*	*Alfanumérico*

Exercício

- Quais os tipos dos dados para as informações abaixo?

- Idade de uma pessoa
- Altura de uma pessoa
- Nome de uma pessoa
- Estado civil de uma pessoa

- Código de um produto
- Descrição de um produto
- Preço de um produto
- Quantidade do produto no estoque

Exercício: solução

- Quais os tipos dos dados para as informações abaixo?

• Idade de uma pessoa	*Tipo inteiro*
• Altura de uma pessoa	*Tipo real*
• Nome de uma pessoa	*Tipo caractere*
• Estado civil de uma pessoa	*Tipo caractere*
• Código de um produto	*Tipo caractere*
• Descrição de um produto	*Tipo caractere*
• Preço de um produto	*Tipo real*
• Quantidade do produto no estoque	*Tipo inteiro*

Variáveis

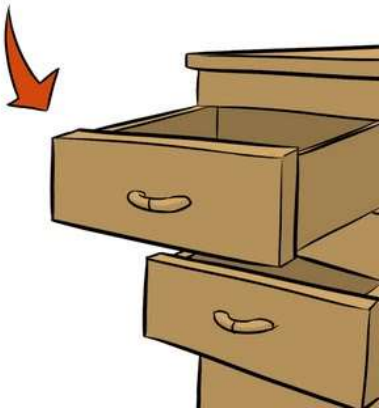
- Essencialmente, programar um computador para executar uma dada tarefa é estabelecer regras de manipulação de informações na sua memória através de uma seqüência de comandos.

- A memória principal funciona como um armário de gavetas, cuja configuração varia de programa para programa.

- Cada programa estabelece o número de gavetas e as gavetas possuem nome, endereço e capacidade de armazenamento diferentes.

- Toda variável x tem um endereço na memória que pode ser acessado com $\&x$.

Nome
Tamanho
Posição (endereço)

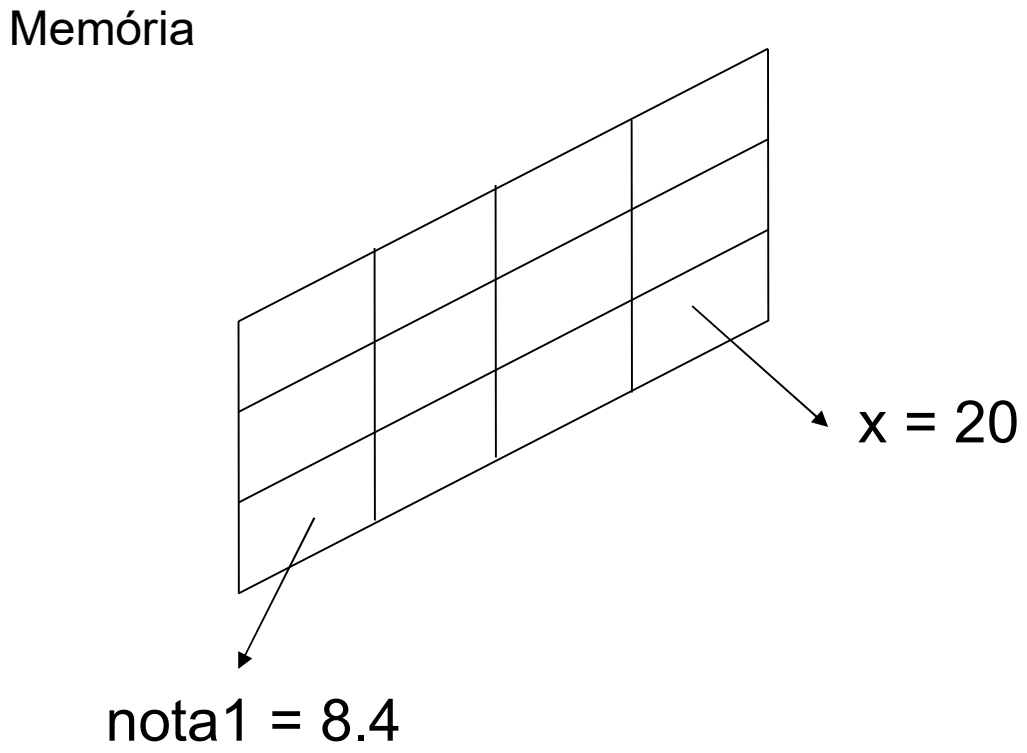


Variáveis

- Dizemos que determinado dado é uma variável quando este pode sofrer alguma alteração, ou seja, ele pode variar

- Em programação, uma variável é um objeto (uma posição localizada na memória) capaz de reter e representar um valor ou expressão. Enquanto as variáveis só "existem" em tempo de execução, elas são associadas a "nomes", chamados identificadores, durante o tempo de desenvolvimento (*Wikipédia*)

Variáveis



Variáveis

- Sintaxe para declaração de uma variável em *pseudocódigo*
  - declarar
    <nome da variável> <tipo de dado da variável> ;
- Exemplo
  - declarar x numérico_inteiro;

Variáveis em C

- Sintaxe para declaração de uma variável em *C*
  - <tipo de dado da variável> <nome da variável> ;
- Exemplo
  - int x;
- Há a possibilidade da declaração de várias variáveis numa mesma linha, basta separá-las por vírgula
- Há a possibilidade de atribuir um valor à variável no momento da declaração da mesma

Atribuição

- O operador “=“ é usado para indicar uma atribuição de valor à variável.
- **Exemplos**:

```
int main {
      int a;
      unsigned int b;
      short c;
      unsigned short d;
      float e;
      double f;
      char g;
      unsigned char h;
      a = 10; /* correto */
      b = -6; /* errado */
      c = 100000; /* errado */
      d = 33000; /* certo */
      e = -80000.657; /* certo */
      f = 30 /* errado */
      g = 'a'; /* certo */
      g = a; /* errado, a menos que ``a'' fosse do tipo char */
      h = 200; /* certo */
      h = 'B'; /* certo */
}
```

Constantes

- Dizemos que determinado dado é uma constante quando este não sofre nenhuma alteração, ou seja, ele é fixo

- Sintaxe para declaração de uma constante em *pseudocódigo*

  – declarar
    constante <nome da constante> ← <valor da constante> <tipo de dado da constante> ;

- Exemplo
  – declarar constante pi ← 3.14 numérico_real;

Constantes em C

- Constantes são usadas para armazenar números e caracteres. Porém, não podemos modificar o conteúdo de uma constante após sua atribuição.
  Exemplos:

```
int main() {
  const float PI = 3.1415926536;
  const char *MSG = "O Conteúdo de a eh ";
  const float Epsilon = 1E-05;
  float dois_PI;
  dois_PI = 2*PI; /* atribui 6.2831853072 para "a" */
  printf("%s %5.2f\n",MSG,dois_PI); /* imprime "O conteúdo de a é 6.28" na tela */
}
```

- Outra forma (não se trata de constante verdadeira)

```
#define PI 3.1415926536 /* atribui 3.1415926536 para PI */
#define MSG "O Conteúdo de a é " /* atribui o texto para MSG */
#define Epsilon 1E-05 /* atribui 0.00001 para Epsilon */
int main() {
  float dois_PI;
  dois_PI = 2*PI; /* atribui 6.2831853072 para "a" */
  printf("%s %5.2f\n",MSG, dois_PI); /* imprime "O conteúdo de a é 6.28" na tela */
}
```

Vetores e Matrizes

Tipos de Dados

- Sistemas de computação têm por finalidade a manipulação de dados
- Um algoritmo, basicamente, consiste na entrada de dados, seu processamento e a saída dos dados computados

- Alguns tipos de dados são considerados básicos, ou primitivos:
  - **Dados Numéricos**: representam valores numéricos diversos
    - Ex: 4, 5.5, 6.1E10, …
  - **Dados Alfanuméricos**: representam valores alfabéticos, numéricos, sinais, símbolos e caracteres
    - Ex: “salário”, “e23”, “%^#”, …
  - **Dados Lógicos**: associados a valores lógicos (*True* ou *False*)
    - Ex: *true*, *false*

Tipos de Dados Compostos

- Dois outros tipos de dados, um pouco mais sofisticado, são extremamente úteis: matrizes e vetores

- Tratam-se de variáveis compostas homogêneas
  - O conteúdo é sempre do mesmo tipo

- Correspondem a posições de memória contíguas
  - Identificadas por um mesmo nome
  - Individualizadas por índices

- Se NOTA contém os seguintes valores:

NOTA	60	70	90	55	91	47	74	86
	1	2	3	4	5	6	7	8

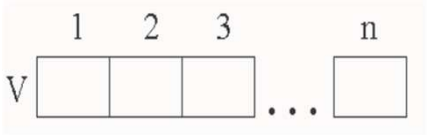
então, NOTA[3] faz referência ao terceiro elemento do conjunto (90)

Vetores

- Também chamado arranjo (array), tipo composto ou agregado
- É uma estrutura de dados homogênea e de acesso aleatório
  - Homogênea: contém elementos de um mesmo tipo;
  - Acesso aleatório: todos seus elementos são igualmente acessíveis a qualquer momento;
- Um elemento específico em uma matriz é acessado através de um índice
- Está disponível na maioria das linguagens
- É usado como base para estruturas de dados mais complexas
- Considere que o próprio acesso a memória do sistema é linear como um vetor

Vetores

- Elementos de um vetor são referenciados com um mesmo *nome*, mas diferenciados por um único *índice*



V = V(1), V(2), …, V(n), onde V(i), com 1 ≤ i ≤ n, é o i-ésimo elemento de V

- Sintaxe: `tipo nome[tamanho];`

- Exemplo:

```
void main(){
  int x[10];  /*reserva 10 elementos inteiros*/
  int i;
  for (i=0; i <10; i++){
    x[i] = i;
  }
}
```

Matrizes

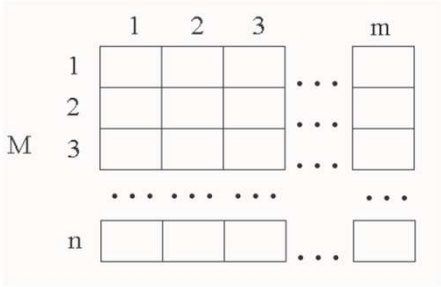
- Arranjo (array) multidimensional. Normalmente, bidimensional;
- Assim como vetores, trata-se de uma estrutura de dados homogênea e de acesso aleatório;
- Necessita de dois ou mais índices
- Também figura na maioria das linguagens;

Matrizes

- Arranjo de várias dimensões de dados do tipo básico, com um mesmo *nome*, mas diferenciado por um *índice* para cada dimensão.

```
    M(1,1)  M(1,2) … M(1,m)
M = M(2,1)  M(2,2) … M(2,m)
    …       …      … …
    M(n,1)  M(n,2) … M(n,m)
```

onde M(*i*,*j*) representa o elemento da *i*-ésima linha e *j*-ésima coluna



- Sintaxe:

```
tipo NomeMatriz[MAX_LINHAS][MAX_COLUNAS];
```

- Exemplo:

```
m[0][1] = 10;
m[1][0] = 20;
printf("Segundo valor de M: %d", m[1][0]);
```

Matrizes

- Forma comum de acessar elementos de uma matriz é por meio de um *loop* duplo, que varre todos os componentes da matriz

```
for (i=0;i<10;i++) {       // percorre a linha
    for (j=0;j<10;j++) {   // percorre a coluna
            a[i][j] = 0;
    }
}
```

- O exemplo acima colocará o valor 0 (zero) para toda a matriz, percorrendo todas as colunas de todas as linhas.

Práticas:

1. Escrever programa que declare um vetor de 100 elementos numéricos, preenchido aleatoriamente com valores entre 1 e 1000. Depois, deve receber um valor do usuário e verificar se existem elementos iguais a esse valor no vetor. Se existir, escrever as posições em que estão armazenados.

2. Escrever programa que declare duas matrizes A e B, inteiras e bidimensionais 5x5, preenchidas aleatoriamente com valores entre 0 e 10. Depois:
    a. Imprimir as matrizes
    b. Executar a multiplicacao de A por B
    c. Imprimir a matriz resultante

Registros

Estrutura de Dados Heterogênea

- Como vimos, temos 4 tipos básicos de dados simples: reais, inteiros, literais e lógicos.

- Podemos usar tais tipos primitivos para definir novos tipos:
  - Agrupam conjunto de dados homogêneos (mesmo tipo) sob um único nome (como os vetores)

  - Agrupam dados heterogêneos (tipos diferentes): Registros

Registros

- Correspondem a conjuntos de posições de memória conhecidos por um mesmo nome, mas com identificadores associados a cada conjunto: membros ou componentes

- Geralmente, todos os membros são logicamente relacionados;
- Sintaxe de criação de Registros em C:

```
struct <nome_do_registro> {
   <componentes_do_registro>
};
```

  - *<nome_do_registro>*: escolhido pelo programador e considerado um novo tipo de dados
  - *<componentes_do_registro>*: declaração das variáveis-membro deste registro, baseada em tipos já existentes

- Um registro pode ser definido em função de outros registros já definidos

Registros

Definindo uma Estrutura

```
struct Endereco {
        char nome[30];
        char rua[50];
        char bairro[20];
        char cidade[20];
        char estado[2];
        int cep;
};
```

Registros

- Neste ponto do código (código anterior), nenhuma variável foi de fato declarada. Apenas a forma dos dados foi definida;

Declarando Variáveis

- Para declarar uma variável com essa estrutura (Endereco), escreva:

```
struct Endereco end_info;
```

- Isso declara uma variável do tipo estrutura *Endereco* chamada *end_info*;

Registros

- Você também pode declarar uma ou mais variáveis enquanto a estrutura é definida. Por exemplo:

```
struct Endereco {
                    char nome[30];
                    char rua[50];
                    char bairro[20];
                    char cidade[20];
                    char estado[2];
                    int cep;
                  } end_info, binfo, cinfo;
```

- define uma estrutura chamada Endereco e declara as variáveis end_info, binfo e cinfo desse tipo;

Registros

Acesso aos membros (ou campos) de uma estrutura

- Elementos individuais de estruturas são referenciados através do operador ponto. Por exemplo;

```
end_info.cep = 130270410;
```

  o nome da variável estrutura seguido por um ponto e pelo nome do elemento referencia esse elemento individual da estrutura;

- Forma Geral: `nome_da_estrutura.nome_do_elemento;`

- Imprimindo o CEP na tela: `printf("%d", end_info.cep);`

Registros

- Um membro (ou campo) de uma estrutura pode ser outra estrutura:

```
struct ENDERECO{
        char rua[30];
        int numero;
        char bairro[30];
        char cidade[30];
        char estado[2];
        char cep[10];
};

struct PESSOA{
        char nome[30];
        char sobrenome[30];
        struct ENDERECO end;
        int idade;
        char rg[15];
        float salario;
};
```

```
void main() {
 ....
 struct PESSOA pess;
 ....
 strcpy(pess.end.rua,
        "Barão de Itapura");
 pess.end.numero = 189;
 strcpy(pess.end.estado, "SP");
 ...
}
```

Funções e Procedimentos

Programação Modular

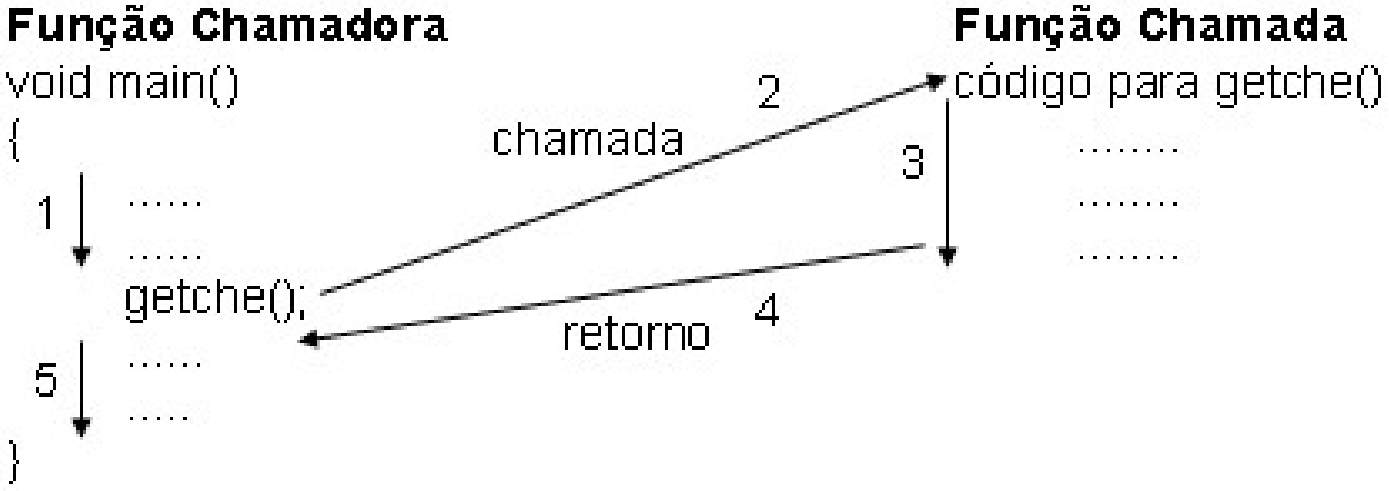
- De acordo com o paradigma da programação estruturada, a escrita de algoritmos (e programas) deve ser baseada no
  - desenho modular dos mesmos
  - passando-se depois a um refinamento gradual

- A modularidade permite entre outros aspectos:
  - Criar diferentes camadas de abstração do programa;
  - Reduzir os custos do desenvolvimento de software e correção de erros;
  - Reduzir o número de erros emergentes durante a codificação;
  - Re-utilização de código de forma mais simples;

- A modularidade pode ser conseguida através da utilização de *sub-rotinas*: funções e procedimentos.

Chamada de Subrotinas

- Uma subrotina é um bloco de código associado a um nome, que pode ser chamado sob demanda a partir de outros pontos do programa



Tipos de Subrotinas

- Na programação estruturada são normalmente definidos dois tipos de sub-rotinas: as funções e os procedimentos

- Função é um tipo e subrotina cujo funcionamento assemelha-se ao de uma função matemática: uma função sempre retorna um valor, como resultado de seu processamento

- Procedimento não retorna um valor após seu processamento, servindo principalmente como um bloco de execução

Funções

- Uma função é definida por um *nome* (*nomeFuncao*), uma *lista de parâmetros*, constituída por zero ou mais variáveis passadas à função, um tipo de retorno e um corpo

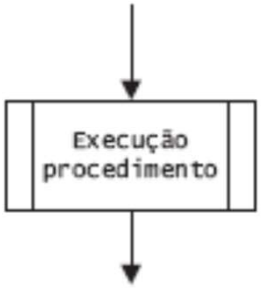

```
<tipo de retorno> nomeFuncao
(listaParametros)
inicio
    <corpo>
    <retorno>
fim
```

Exemplo em C:

```
long potencia (int base, int expoente){
  /* Variavel de resultado */
  long resultado = 1; int i;
  /* Calcula potencia atraves de multip. Sucessivas */
  for (i = 1; I <= expoente; i++) {
    resultado *= base;
  }
  /* Retorna valor calculado */
  return resultado;
}
```

Procedimento

- Um procedimento é definido por um *nome* (*nomeProc*), uma *lista de parâmetros*, constituída por zero ou mais variáveis e um corpo




```
<nulo> nomeProc (listaParametros)
inicio
    <corpo>
fim
```

Exemplo em C:

```
void numeroInvertido (int numero){
  while (numero > 0) {
    /* Calcula algarismo + a direita atraves de divisao
       inteira por 10 */
    int algarismo = numero % 10;
    printf ("%i", algarismo);
    /* Trunca algarismo a direita */
    numero = (numero - algarismo) / 10;
  }
}
```

Práticas:

3. Escreva uma função em C, que receba um **número inteiro**, como parâmetro, e devolva o maior algarismo contido nesse número (não trate a entrada como *string*).

Parâmetros

Passagem de Parâmetros

- Dados são passados pelo algoritmo principal (ou outro subalgoritmo) à subrotina, ou retornados por este ao primeiro, por meio de parâmetros

  - Parâmetros formais são os nomes simbólicos (variáveis) introduzidos no cabeçalho das subrotinas, utilizados na definição dos seus parâmetros. Dentro da subrotina trabalha-se com estes nomes da mesma forma como se trabalha com variáveis locais.

    Ex:
    ```
    float calcPotencia(int base, int expoente) {…}
    ```

  - Parâmetros reais são aqueles que substituem os parâmetros formais na chamada de uma subrotina. Os parâmetros formais são úteis somente na definição do subalgoritmo. Os parâmetros reais podem ser diferentes a cada chamada.

    Ex:
    ```
    int a = 2;
    calcPotencia(a, 3);
    ```

Mecanismos de Passagem

- Os parâmetros reais substituem os parâmetros formais no ato da chamada de uma subrotina.

- Esta substituição é denominada passagem de parâmetros e pode se dar por dois mecanismos:

  - Passagem por Valor (cópia): na passagem de parâmetros por valor o parâmetro real é calculado e uma cópia de seu valor é fornecida ao parâmetro formal.
    - Modificações feitas no parâmetro formal não afetam o parâmetro real
    - Parâmetros formais possuem locais de memória exclusivos para que possam armazenar os valores dos parâmetros reais.

  - Passagem por Referência: na passagem de parâmetros por referência não é feita uma reserva de espaço de memória para os parâmetros formais.
    - Espaço de memória ocupado pelos parâmetros reais é compartilhado pelos parâmetros formais correspondentes
    - Eventuais modificações feitas nos parâmetros formais também afetam os parâmetros reais correspondentes

Passagem de Parâmetros em C

- Em C, a passagem de parâmetro é normalmente feita por valor:
  - Passagem por Valor (cópia):
    - Declaração da função:

      ```
      int minhaFuncao(int x){...}
      ```
    - Chamada:

      ```
      int a = 23;
      minhaFuncao(a)
      ```
    - Uma cópia do valor do parâmetro real “a” é atribuida ao parâmetro formal “x”
  - Passagem por Referência: em C, a passagem por referência é feita através do uso de ponteiros
    - Declaração:

      ```
      int minhaFuncao(int *x) {…}
      ```
    - Chamada:

      ```
      int a = 23;
      minhaFuncao(&a);
      ```
    - O endereço do parâmetro real “a” é passado para o ponteiro “x”. Logo, o parâmetro formal “x” referencia a mesma posição de memória de “a”. Por isso, qualquer alteração feita em “x” é refletida em “a”.

Práticas:

4. Modularize o código do Exercício 2. Sugira e implemente uma alternativa para evitar a repetição do código que imprime as matrizes.

5. Escreva uma função em C que receba um número inteiro “i” e uma matriz 3x3 como parâmetros. A função deve calcular a multiplicação “m” dos elementos da diagonal principal dessa matriz. Deve atualizar a matriz original, multiplicando todos os seus elementos por “i”. O valor “m” calculado deve ser retornado.