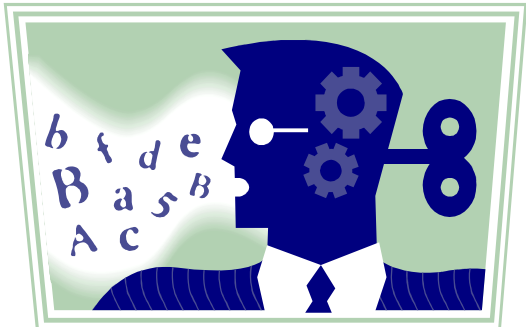
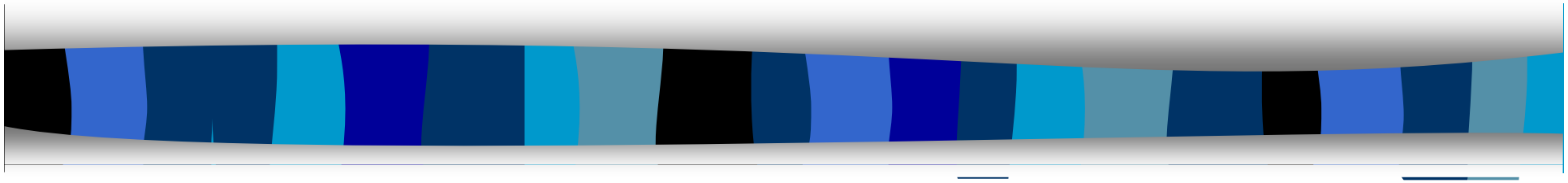
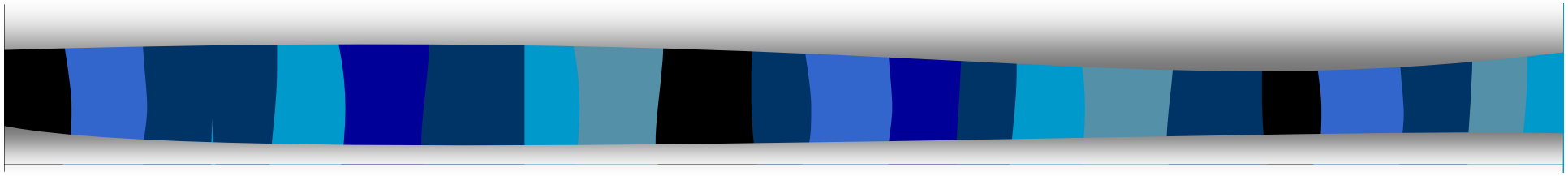




Revisão de APC II (cont)



Funções e Procedimentos

Programação Modular

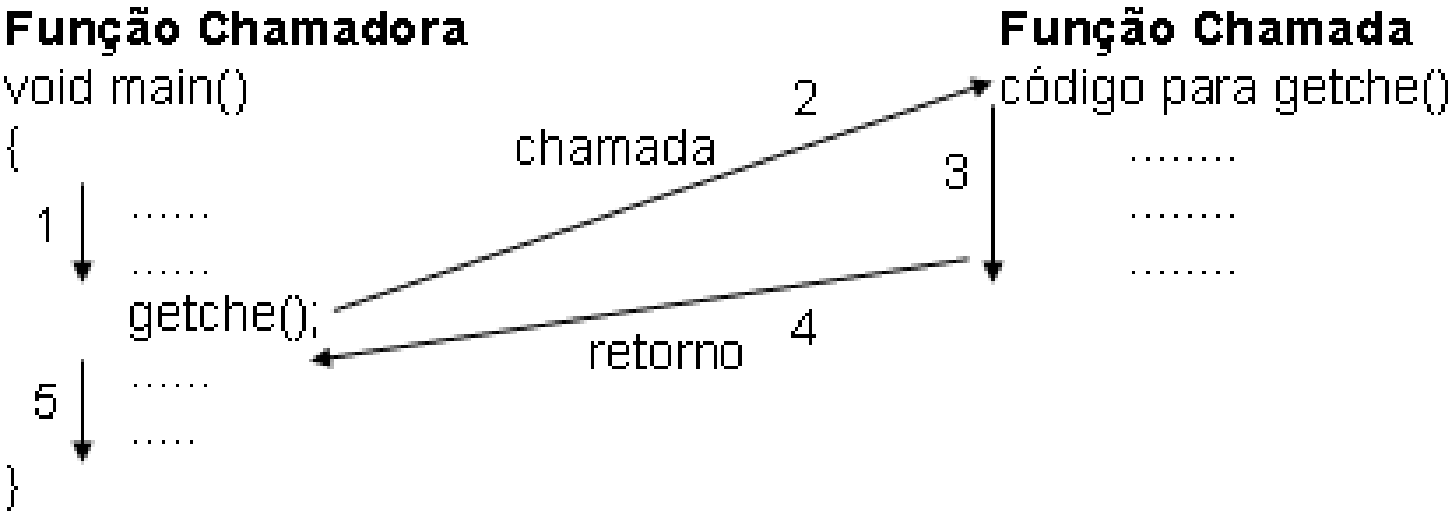
- De acordo com o paradigma da programação estruturada, a escrita de algoritmos (e programas) deve ser baseada no
  - desenho modular dos mesmos
  - passando-se depois a um refinamento gradual

- 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

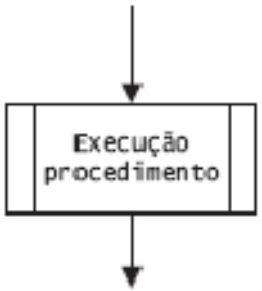

```
<tipo de retorno> nomeFuncao
(listaParametros)
inicio
    <corpo>
    <retorno>
fim
```

Exemplo em C:

```
long potencia (int base, int expoente){
  /* Variavel de resultado */
  long resultado = 1;
  /* Calcula potencia atraves de multip. Sucessivas */
  for (int 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



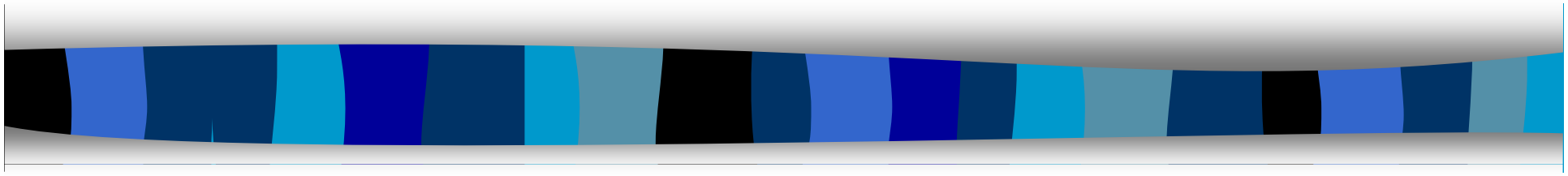

```
<nulo> nomeProc (listaParametros)
inicio
    <corpo>
fim
```

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;
  }
}
```

Exercício:

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.

  ```
  int a = 2;
  ```
  Ex: `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

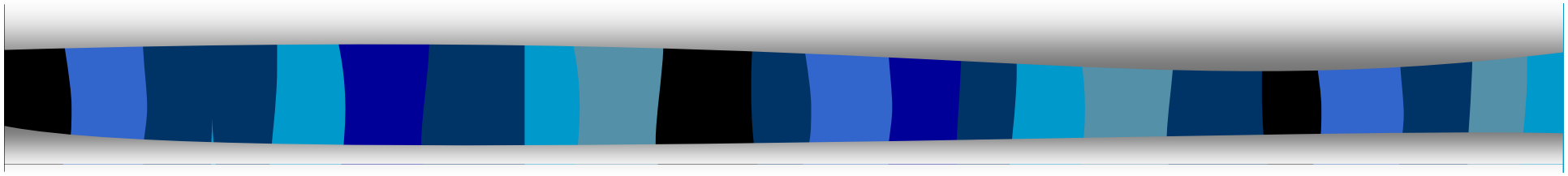
Passagem de Parâmetros em C

- 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”.

Exercícios:

4. Modularize o código do Exercício 2, apresentado no laboratório anteriormente. 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.



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”);
 ...
}
```