ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES

Disciplina: 113476

Profa. Carla Denise Castanho

Universidade de Brasília – UnB
Instituto de Ciências Exatas – IE
Departamento de Ciência da Computação – CIC

5. SUBALGORITMOS

INTRODUÇÃO

Subproblemas

- Frequentemente, os problemas que precisamos resolver são **complexos**, e requerem algoritmos **grandes** e **difíceis de ler**.
- Nesses casos, é comum acontecerem várias **repetições** de um mesmo conjunto de comandos em partes diferentes do código.
- Isso traz duas consequências ruins, como veremos a seguir...

Subproblemas

- Primeira consequência: dificuldade de **manutenção**!
- Se for necessário **modificar** algum detalhe em um trecho de código que se repete, precisaremos alterar em **todos os lugares** em que ele ocorre!
- Por exemplo: suponha que o cálculo de um imposto precise ser feito em 10 pontos diferentes de um programa – caso a regra de cálculo mude, precisaremos alterá-la 10 vezes, com muitas chances de **cometer erros**!

Subproblemas

- Segunda consequência: dificuldade de **abstração**!
- Várias vezes, um algoritmo pode ser descrito em termos de comandos de mais **alto nível**, abstraindo-se detalhes que não são importantes para entender a **idéia geral**.
- Por exemplo: o comando "*solte os parafusos*" pode ser descrito em termos de instruções mais específicas (encaixe da ferramenta, ângulo, força, número de repetições), porém, não é necessário esse **nível de detalhe** para entender o algoritmo de troca de pneus.
- Se esses detalhes fossem incluídos no **corpo** do algoritmo, seriam repetidos várias vezes e apenas atrapalhariam o entendimento, pois não são relevantes para o **fluxo principal** da solução.

Subproblemas

- Chamamos esses conjuntos de comandos auxiliares de **subproblemas**, i.e., eles existem para resolver problemas menores ou tarefas específicas, que devem ser solucionados para atingir o objetivo do algoritmo.
- Imagine uma fábrica de automóveis: ela projeta e monta os carros, mas não fabrica os pneus e os vidros, por exemplo, comprando-os prontos ou produzindo-os em outro local; isso acontece por que seria muito complicado manter uma linha de produção de carros, pneus e vidros, e gerir tudo isso em um mesmo lugar.

Modularização

- Em programação, também podemos "**terceirizar**" o trabalho, resolvendo os subproblemas em **trechos separados** de código, chamados ***módulos***.
- A idéia é **dividir** para conquistar, definindo os algoritmos em termos de **subalgoritmos** menores e de mais fácil compreensão, ou seja, transformar um problema difícil em vários problemas mais fáceis, e no final juntar tudo para formar uma solução completa.
- A maioria das linguagens de programação suporta esse conceito e define as regras formais necessárias para realizar tal divisão. Dependendo da linguagem, subalgoritmos são comumente chamados de **subrotinas** (ex.: Basic, Logo), **procedimentos** (ex.: Pascal), **funções** (ex.: C, Fortran) ou **métodos** (ex.: C++, Java).
- Em APC, adotaremos uma sintaxe menos rígida, *i.e.*, uma **convenção**, parecida com o exemplo a seguir...

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

SUBALGORITMO (FUNÇÃO)

ALGORITMO PRINCIPAL

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

Toda função deve ter um nome único.

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

Toda função deve ter um nome único.

Uma função pode retornar valores, e, portanto, deve ter um tipo de retorno.

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

Toda função deve ter um nome único.

Uma função pode retornar valores, e, portanto, deve ter um tipo de retorno.

Uma função pode receber parâmetros, que também devem ter um nome e um tipo.

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

Toda função deve ter um nome único.

Uma função pode retornar valores, e, portanto, deve ter um tipo de retorno.

Uma função pode receber parâmetros, que também devem ter um nome e um tipo.

Utilize a palavra-chave retorne para sair da função e retornar um valor a quem a chamou.

Definindo Subalgoritmos

- Ex.: Faça um faça o algoritmo que leia um número inteiro digitado pelo usuário e mostre na tela seu quadrado, que deve ser calculado por um subalgoritmo (função).

Exemplo - Definindo um Subalgoritmo

```
Algoritmo Modular

Função Quadrado(a : inteiro) : inteiro
Início
    retorne a * a
Fim

Variáveis
    n : inteiro
Início
    Escreva ("Informe um número:")
    Leia (n)
    Escreva (n, " ao quadrado é ", Quadrado(n))
Fim
```

Toda função deve ter um nome único.

Uma função pode retornar valores, e, portanto, deve ter um tipo de retorno.

Uma função pode receber parâmetros, que também devem ter um nome e um tipo.

Utilize a palavra-chave retorne para sair da função e retornar um valor a quem a chamou.

Observe que a função é chamada pelo nome, recebendo os parâmetros entre parênteses.

Definindo Subalgoritmos

- Portanto, nossa convenção para definição de subalgoritmos será:

Sintaxe para declarar um Subalgoritmo

```
Função <nome> ([<lista de parâmetros>]) : [<tipo de retorno>]
Início
    <comandos>
    [retorne <expressão>]
Fim
```

* expressões entre chaves podem aparecer ou não, conforme explicado abaixo

- O **nome** da função deve ser **único**.
- A **lista de parâmetros** deve ser da forma:

```
<nome param1> : <tipo param1>, <nome param2> : <tipo param2>, ...
```

- Uma função pode não receber nenhum parâmetro, nesse caso, terá uma **lista vazia**: `()`.
- Se a função retornar algum valor, deve-se definir um **tipo de retorno** e é obrigatória a existência do comando **`retorne`** *<expressão>*.
- Observe que toda função também tem um **bloco** com **Início** e **Fim**, para delimitá-la.

Definindo Subalgoritmos

- Quando nós chamamos uma função passando variáveis, os **nomes das variáveis não têm nenhuma relação com os nomes dos parâmetros!**
- Os nomes dos parâmetros só importam dentro da função!
- Quem chama a função pode passar qualquer valor: uma **expressão**, uma **variável**, uma **constante**, o resultado de outra **chamada de função**, etc... É necessário respeitar apenas a **ordem** e o **tipo** dos parâmetros:

Definindo Subalgoritmos

- Quando nós chamamos uma função passando variáveis, os **nomes das variáveis não têm nenhuma relação com os nomes dos parâmetros!**
- Os nomes dos parâmetros só importam dentro da função!
- Quem chama a função pode passar qualquer valor: uma **expressão**, uma **variável**, uma **constante**, o resultado de outra **chamada de função**, etc... É necessário respeitar apenas a **ordem** e o **tipo** dos parâmetros:

Chamando Subalgoritmos

```
Algoritmo ChamaFuncao

Função TrataData(dia : inteiro, mes : inteiro, ano : inteiro)
Início
    ...
Fim

Variáveis
    d, m, a : inteiro
Início
    ...
    TrataData(d, m, a)
    ...
Fim
```

Observe: o nome das variáveis e os parâmetros da função são diferentes, não têm relação entre si! Mas seria errado chamar TrataData(a, m, d) pois a função espera como primeiro parâmetro o dia, como segundo, o mês e por fim, como terceiro, o ano. Além disso, temos que passar valores inteiros, não poderíamos passar um float ou uma string, por exemplo.

Definindo Subalgoritmos

- Mas, e se meu subalgoritmo também precisar de **variáveis**?
- Simples! Basta declará-las antes do bloco:

Definindo Subalgoritmos

- Mas, e se meu subalgoritmo também precisar de **variáveis**?
- Simples! Basta declará-las antes do bloco:

Sintaxe para declarar um Subalgoritmo

```
Função <nome> ([<lista de parâmetros>]) : [<tipo de retorno>]
Variáveis
    <nome1> : <tipo1>
    <nome2> : <tipo2>
    ...
Início
    <comandos>
    [retorne <expressão>]
Fim
```

Definindo Subalgoritmos

- Mas, e se meu subalgoritmo também precisar de **variáveis**?
- Simples! Basta declará-las antes do bloco:

Sintaxe para declarar um Subalgoritmo

```
Função <nome> ([<lista de parâmetros>]) : [<tipo de retorno>]
Variáveis
    <nome1> : <tipo1>
    <nome2> : <tipo2>
    ...
Início
    <comandos>
    [retorne <expressão>]
Fim
```

- Importante! Em uma função, os **parâmetros também são considerados variáveis** e seguem as mesmas regras! Por exemplo, não podemos repetir nomes.

Exemplo

- Primeiro, um exemplo simples: uma função SEM parâmetros e SEM nenhum retorno.

```
Algoritmo Cumprimento

Função LimpaTela()
Variáveis
    i : inteiro
Início
    Para i ← 1 até 50 faça
        Escreva (“  ”)
    Fim-Para
Fim

Variáveis
    nome : string
Início
    Escreva (“Qual a sua graça?”)
    Leia (nome)
    LimpaTela()
    Escreva (“Diga lá, ”, nome, “! Você programa sempre por aqui?”)
Fim
```

Exemplo

- Primeiro, um exemplo simples: uma função COM parâmetros e SEM nenhum retorno.

Exemplo - Função com parâmetros e sem retorno

```
Algoritmo Cumprimento

Função LimpaTela(linhas:inteiro)
Variáveis
    i : inteiro
Início
    Para i ← 1 até linhas faça
        Escreva (“  ”)
    Fim-Para
Fim

Variáveis
    nome : string
Início
    Escreva (“Qual a sua graça?”)
    Leia (nome)
    LimpaTela(10)
    Escreva (“Diga lá, ”, nome, “! Você programa sempre por aqui?”)
Fim
```

Exemplo

- ... e uma que retorna valores.

Exemplo - Função com retorno

```
Algoritmo ValidaEntrada

Função LePositivo() : inteiro
Variáveis
    n : inteiro
Início
    Escreva ("Informe um número inteiro positivo:")
    Leia (n)
    Enquanto n <= 0 faça
        Escreva ("ERRO! O número deve ser positivo! Informe novamente:")
        Leia (n)
    Fim-Enquanto
    retorne n
Fim

Variáveis
    i, vezes : inteiro
Início
    vezes ← LePositivo()
    Para i ← 1 até vezes faça
        Escreva ("Olá, submundo!")
    Fim-Para
Fim
```

Exemplo

- Agora um exemplo completo!

Exemplo - Fatorial

```
Algoritmo FatorialModular

Função Fatorial(n : inteiro) : inteiro
Variáveis
    fat : inteiro
Início
    fat ← 1
    Enquanto n > 1 faça
        fat ← fat * n
        n ← n - 1
    Fim-Enquanto
    retorne fat
Fim

Variáveis
    num : inteiro
Início
    Escreva ("Informe um número inteiro:")
    Leia (num)
    Escreva ("O fatorial de ", num, " é ", Fatorial(num), ".")
Fim
```

A instrução “retorne”

- Quando uma função encontra a instrução `retorne`, sua execução é imediatamente interrompida e o programa **volta ao ponto em que a função foi chamada**. Se um valor de retorno for passado, esse valor retorna para quem chamou a função.
- É importante que o valor ou expressão de retorno seja **compatível** com o **tipo de retorno** da função.
- Uma mesma função pode ter várias instruções `retorne` em pontos diferentes. Se qualquer uma delas for atingida, a função retorna imediatamente.

Exemplo

Exemplo - Mais de uma instrução "retorne"

```
Algoritmo ExemploRetorne

Função Maior(a : inteiro, b : inteiro) : inteiro
Início
    Se a > b
        retorne a
    Senão
        retorne b
    Fim-Se
Fim

Variáveis
    n1, n2 : inteiro
Início
    Escreva ("Informe dois números: ")
    Leia (n1, n2)
    Escreva ("O maior é: ", Maior(n1, n2), ".")
Fim
```

Funções na Linguagem C

- Mas como definimos funções em C?

Funções na Linguagem C

- Mas como definimos funções em C?

Exemplo - Funções em C

```
#include <stdio.h>

int maior (int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}

int main () {
    int n1, n2;
    printf(”Informe dois números: \n”);
    scanf("%d %d", &n1, &n2);
    printf("O maior eh: %d.\n", maior(n1, n2));

    return 0;
}
```

Funções na Linguagem C

- Pelo exemplo, podemos ver que uma função pode ser declarada antes da ***main***, e que seu formato é muito parecido com a *main*. De fato, a *main* é uma função!
- Formalmente, uma função C tem a seguinte sintaxe:

Funções em C - Sintaxe

```
<tipo de retorno> <nome> ([<lista de parâmetros>]) {
    <instruções>
}
```

A palavra-chave “void”

- Em C, o tipo de retorno da função é obrigatório.
- Caso uma função não retorne nenhum valor, seu tipo deve ser definido como `void` (vazio, em inglês):

Funções C sem retorno

```
void LimpaTela(int linhas) {
    for  (i=1; i<=linhas; i++)
       printf(”\n”);
}
```

- Da mesma forma que em pseudo-código, a lista de parâmetros pode ser vazia. No entanto, você também pode utilizar `void` para indicar explicitamente este fato:

Funções C sem parâmetros

```
int le_positivo () {
    int n;

    printf(”Informe um número: \n”);
    scanf("%d", &n);
    while (n <= 0)
        scanf("%d", &n);
    return n;
}
```

Funções C sem parâmetros

```
int le_positivo (void) {
    int n;

    printf(”Informe um número: \n”);
    scanf("%d", &n);
    while (n <= 0)
        scanf("%d", &n);
    return n;
}
```

Funções na Linguagem C

- Você já conhece várias funções pré-definidas da Linguagem C...
  - Como vimos, `main` é uma função, mas ela é especial:
    - O compilador espera que exista exatamente uma função chamada *main*, e ela será sempre a primeira função a ser executada no programa;
    - ela retorna um valor inteiro, que é lido pelo sistema operacional – por convenção, se não houver nenhum problema, *main* deve retornar zero, caso contrário, deve retornar um valor diferente de zero, um código de erro, por exemplo.
  - `printf` e `scanf` também são funções! A função *printf* retorna um valor inteiro indicando o total de caracteres impressos na tela, *scanf* retorna o total de elementos efetivamente lidos. Normalmente, esses valores não são utilizados pelo programador, por isso, são ignorados pelo compilador.
  - Todos os comandos ativados por meio da diretiva `#include` são, na verdade, funções! Exemplos conhecidos: `sqrt`, `pow`, `getchar`, etc...
  - Quer conhecer todas as funções pré-definidas da Linguagem C? Acesse: http://www.cplusplus.com/reference/clibrary/

Subalgoritmos – Exercícios

1. Faça um algoritmo que leia dois pontos no plano cartesiano e calcule o coeficiente angular da reta que passa por estes pontos. Seu algoritmo deve criar uma função auxiliar que RECEBE 4 parâmetros reais – $x0, y0, x1, y1$ – que são as coordenadas dos pontos e RETORNA um valor real que é o coeficiente angular da reta, dado por $(y1 - y0) / (x1 - x0)$.
2. Faça um algoritmo que leia um número de pessoas n, e, para cada pessoa, leia sua altura e sexo (M/F), informando, em seguida, seu peso ideal. Seu algoritmo deve criar uma função auxiliar que RECEBE a altura e o sexo de uma pessoa e RETORNA seu peso ideal, dado por $k * h^2$. Onde h é a altura, em metros, e k, quando se tratar de um homem, é 23, quando mulher, 20.
3. Faça um algoritmo com uma função que calcule a potência do número inteiro X elevado ao número inteiro Y, ou seja, X^Y. Seu algoritmo principal deve ler os valores e PASSÁ-LOS para a função, mostrando o resultado RETORNADO pela função em seguida. Valide (mostrando mensagem de erro, se for o caso) os seguintes requisitos: o valor informado para Y deve ser não-negativo, e $X^0 = 1$, desde que $X \neq 0$.