

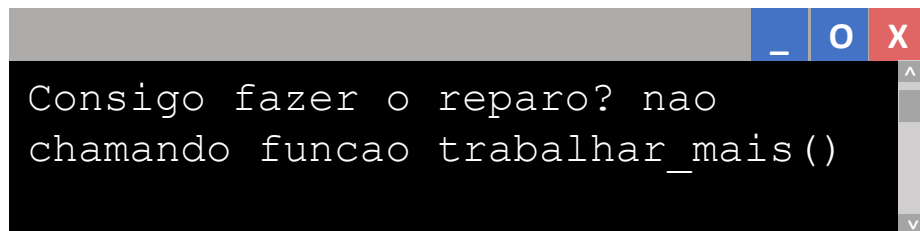
Programação e Desenvolvimento de Software I

Processamento condicional

Prof. Héctor Azpúrua
(slides adaptados do Prof. Pedro Olmo)

Problema 1

- Determine se consigo:
 - Reparar o carro com 400 reais;
 - E com 4999?



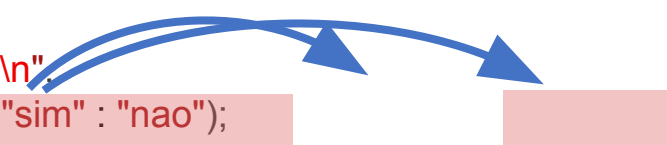
```
Consigo fazer o reparo? nao
chamando funcao trabalhar_mais()
```

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

```
// gcc p1.c -o p1 -lm && ./p1
```

```
int main() {
    float poupanca = 400;
    float orcamento_carro = 1050;
    float diferenca_valor = poupanca - orcamento_carro;
```

```
    printf("Consigo fazer o reparo? %s\n",
           (diferenca_valor >= 0) ? "sim" : "nao");
```



```
    if (diferenca_valor >= 0) {
        // reparar_carro();
        printf("chamando funcao reparar_carro()\n");
    } else {
        // trabalhar_mais();
        printf("chamando funcao trabalhar_mais()\n");
    }
```

```
    return 0;
```

```
}
```

Problema I

- Determine se consigo:
 - Reparar o carro com 400 reais;
 - E com 4999?

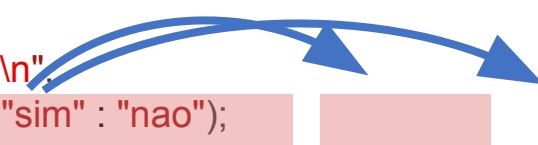
```
Consigo fazer o reparo? sim
chamando funcao reparar_carro()
```

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

```
// gcc p1.c -o p1 -lm && ./p1
```

```
int main() {
    float poupanca = 4999;
    float orcamento_carro = 1050;
    float diferenca_valor = poupanca - orcamento_carro;
```

```
printf("Consigo fazer o reparo? %s\n",
       (diferenca_valor >= 0) ? "sim" : "nao");
```



```
if (diferenca_valor >= 0) {
    // reparar_carro();
    printf("chamando funcao reparar_carro()\n");
} else {
    // trabalhar_mais();
    printf("chamando funcao trabalhar_mais()\n");
}
```

```
return 0;
```

```
}
```

Processamento condicional

- Para executar um processamento condicional, um programa precisa utilizar o comando **if**
- Todo comando **if** requer uma condição.
 - O valor de uma condição pode ser **verdadeiro** ou **falso**
- Em C, não existe um tipo de dados específico para representar valores lógicos (V ou F)

Qualquer valor **diferente de zero** é interpretado como verdadeiro, enquanto **zero é falso**

Operadores relacionais

- Para escrever condições, são utilizados:
 - Os **operadores relacionais** e os **operadores lógicos**

```
int a = 3;  
float x = 1.5;
```

Operador	Significado
----------	-------------

Condição	Valor lógico
----------	--------------

Operadores lógicos

- Os operadores lógicos permitem **combinar várias condições** em uma única expressão lógica

```
int a = 3;  
float x = 1.5;
```

Operador	Significado	Condição	Valor lógico

Operador condicional

- O operador condicional na linguagem C tem a seguinte sintaxe:

`(condição) ? resultado-se-condição-verdadeira : resultado-se-condição-falsa`

- Os resultados podem ser de qualquer tipo (int, float, char, double) e mesmo strings

```
(b != 0) ? a/b : 0;  
(peso <= 75) ? "ok" : "deve emagrecer";
```

Operador condicional

- O operador condicional pode ser usado em atribuições
- Exemplo:

```
float nota1 = 5.0;  
float nota2 = 4.0;  
float media;  
media = ((nota1 >= 3) && (nota2 >= 5)) ?  
    (nota1 + 2 * nota2) / 3 :  
    (nota1 + nota2) / 2;
```

media recebe o valor 4.5

Qual seria o valor de média se:

```
float nota1 = 5.0;  
float nota2 = 6.5;
```


Operador condicional

- No programa **p1.c**, o operador condicional é usado dentro da função `printf`

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

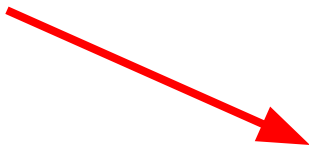
// gcc p1.c -o p1 -lm && ./p1

int main() {
    float poupanca = 400;
    float orcamento_carro = 1050;
    float diferenca_valor = poupanca - orcamento_carro;

    printf("Consigo fazer o reparo? %s\n",
           (diferenca_valor >= 0) ? "sim" : "nao");

    if (diferenca_valor >= 0) {
        // reparar_carro();
        printf("chamando funcao reparar_carro()\n");
    } else {
        // trabalhar_mais();
        printf("chamando funcao trabalhar_mais()\n");
    }

    return 0;
}
```



Atribuição e teste de igualdade

■ Atenção:

- Um erro comum em linguagem C é usar o operador de atribuição (=) em vez do operador relacional (==) em condições que testam igualdade

```
int fator = 3;  
if (fator == 1) {  
    printf("O fator e' unitario\n");  
}
```

```
printf("fator = % d\n", fator);
```

É verdadeiro

```
int fator = 3;  
if (fator = 1) {  
    printf("O fator e' unitario\n");  
}
```

```
printf("fator = % d\n", fator);
```

```
fator = 3
```

```
O fator e' unitario  
fator = 1
```

Problema 2

- Dada uma temperatura em graus centígrados, apresentá-la em graus Fahrenheit
- A fórmula de conversão é:
 - $F = (9 * C + 160) / 5$

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

// gcc p2.c -o p2 -lm && ./p2

int main() {
    float c, f;
    printf("Digite a temperatura em graus C:");
    scanf("%f", &c);

    f = (9 * c + 160) / 5;
    printf("Esta temperatura e %.1f graus F\n", f);
    return 0;
}
```

Problema 2

- Nos programas anteriores, os valores das variáveis eram estabelecidos em operações de atribuição
- **Mas agora, qual é o valor de C?**

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

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

```
// gcc p2.c -o p2 -lm && ./p2
```

```
int main() {
```

```
    float c, f;
```

```
    printf("Digite a temperatura em graus C:");
```

```
    scanf("%f", &c);
```

```
    f = (9 * c + 160) / 5;
```

```
    printf("Esta temperatura e %.1f graus F\n", f);
```

```
    return 0;
```

```
}
```

Leitura de dados

- Uma outra forma de atribuir valores a variáveis é a **leitura de dados**
 - Em C, usa-se a função `scanf`
- Assim como `printf`, a função `scanf` pode ter vários parâmetros, sendo o primeiro uma **string**
 - No caso da função `scanf`, esta string deve conter apenas tags separadas por espaços em branco
 - `"%d %f"`
- Os demais parâmetros da função `scanf` devem ser endereços de variáveis

Leitura de dados

- O que acontece quando o computador executa uma **instrução de leitura de dados**?

- Exemplo:

```
scanf("%f", &c);
```

- A execução do programa é **interrompida**. O computador espera que o usuário digite algum valor e **pressione a tecla Enter**
- Após pressionar **Enter**, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s) no(s) endereço(s) fornecido(s) na função `scanf`

Leitura de dados

- Após pressionar **Enter**, o computador retoma a execução do programa e armazena o(s) valor(es) digitado(s) no(s) endereço(s) fornecido(s) na função `scanf`

```
scanf("%f", &c);
```

↓
[Exemplo: 20 + enter]



Endereço de Memória	Variável Alocada	Valor
10FA0001	C	
10FA0002	aux	
10FA0003		
10FA0004		

Leitura de dados

- O que difere a leitura de dados da operação de atribuição?

Na operação de atribuição, o valor a ser atribuído é definido antes da execução do programa, enquanto numa operação de leitura de dados, o valor atribuído é definido durante a execução.

- Em programação, diz-se que coisas são estáticas quando ocorrem antes do programa executar e dinâmicas quando ocorrem durante a execução

<code>c = 32;</code>		Valor de C estabelecido estaticamente
<code>scanf("%f", &c);</code>		Valor de C estabelecido dinamicamente

Leitura de dados

- Na leitura de dados, o valor digitado pelo usuário deve ser do **mesmo tipo** que a variável
- Com a leitura de dados, a **execução** de um programa pode ser realizada para valores diferentes das variáveis
- Porém, **se o valor da variável é estabelecido de forma estática, para cada valor da variável, é necessário compilar o programa novamente**

Problema 3

- Dadas as idades (tipo **int**) e os pesos (tipo **float**) de duas pessoas, exibir quem é a pessoa mais velha e a sua idade e quem é a pessoa mais leve e o seu peso

```
int main() {  
    int idade1, idade2, maior_idade;  
    int mais_velho, mais_leve;  
    float peso1, peso2, menor_peso;  
  
    printf("Digite idade e peso da pessoa 1:");  
    scanf("%d %f", &idade1, &peso1);  
  
    printf("Digite idade e peso da pessoa 2:");  
    scanf("%d %f", &idade2, &peso2);
```

```
    if (idade1 > idade2) {  
        maior_idade = idade1;  
        mais_velho = 1;  
    } else {  
        maior_idade = idade2;  
        mais_velho = 2;  
    }  
}
```

```
    if (peso1 < peso2) {  
        menor_peso = peso1;  
        mais_leve = 1;  
    } else {  
        menor_peso = peso2;  
        mais_leve = 2;  
    }  
}
```

```
    printf("Maior idade=%d (pessoa %d)\n",  
           maior_idade, mais_velho);  
    printf("Menor peso=%f (pessoa %d)\n",  
           menor_peso, mais_leve);  
    return 0;
```

```
}
```

Comando if-else

- Todo comando if requer uma condição que pode ser verdadeira ou falsa
- **Caso a condição seja verdadeira**, o comando **if** executa um conjunto de instruções, podendo deixar de executar um outro conjunto alternativo
- Quando existe um conjunto de instruções a ser executado, caso o valor da condição seja falso, utiliza-se o comando **if-else**

Comando if-else

- Exemplo:

```
if (diferenca_valor >= 0) {  
    // reparar_carro();  
    printf("chamando funcao reparar_carro()\n");  
} else {  
    // trabalhar_mais();  
    printf("chamando funcao trabalhar_mais()\n");  
}
```

- Um conjunto de instruções começa com o símbolo { e termina com }
- Caso o conjunto tenha **apenas uma instrução** as chaves são opcionais!

Comando if-else

- Caso o conjunto tenha **apenas uma instrução** as chaves são opcionais:

```
if (diferenca_valor >= 0)
    printf("chamando funcao reparar_carro()\n");
else
    printf("chamando funcao trabalhar_mais()\n");
```

- **Importante:**

- **Qualquer instrução** pode fazer parte de um conjunto de instruções, inclusive um comando **if** ou um comando **if-else**

Comandos if-else interrelacionados

- Exemplo:
 - Imagine uma função que recebe como parâmetro um inteiro representando o número de um mês e retorna o número de dias deste mês (considere que fevereiro tem sempre 28 dias)

```
int dias_do_mes(int mes) {  
    int dias;  
    if (mes == 1) dias = 31;  
    if (mes == 2) dias = 28;  
    if (mes == 3) dias = 31;  
    if (mes == 4) dias = 30;  
    if (mes == 5) dias = 31;  
    if (mes == 6) dias = 30;  
    if (mes == 7) dias = 31;  
    if (mes == 8) dias = 31;  
    if (mes == 9) dias = 30;  
    if (mes == 10) dias = 31;  
    if (mes == 11) dias = 30;  
    if (mes == 12) dias = 31;  
    else  
        dias = 0;  
    return dias;  
}
```

Comandos if-else interrelacionados

- Uma outra forma de escrever esta função, mas ainda com comandos if-else inter-relacionados é:

```
int dias_do_mes2(int mes) {  
    int dias;  
    if ((mes == 1) || (mes == 3) ||  
        (mes == 5) || (mes == 7) ||  
        (mes == 8) || (mes == 10) || (mes == 12))  
        dias = 31;  
    else if (mes == 2)  
        dias = 28;  
    else if ((mes == 4) || (mes == 6) ||  
        (mes == 9) || (mes == 11))  
        dias = 30;  
    else  
        dias = 0;  
  
    return dias;  
}
```

Comando switch

- A demanda por comandos **if-else inter-relacionados** é muito comum em programação
- Assim, a linguagem C disponibiliza um comando especial para tais situações: **switch**.

```
switch (expressão)
{
    case constante-1:
        comandos-1;
    case constante-2:
        comandos-2;
    ...
    default:
        comandos-n;
}
```


Comando switch

- Com o comando switch, a função **dias_do_mes** pode ser reescrita como:

```
int dias_do_mes3(int mes) {  
    int dias;  
    switch (mes) {  
        case 1:  
        case 3:  
        case 5:  
        case 7:  
        case 8:  
        case 10:  
        case 12:  
            dias = 1;  
            break;  
        case 2:  
            dias = 28;  
            break;  
        case 4:  
        case 6:  
        case 9:  
        case 11:  
            dias = 30;  
            break;  
        default:  
            dias = 0;  
    }  
  
    return dias;  
}
```

Comando switch

- Este comando permite que, de acordo com o valor de uma expressão, seja executado um ou mais comandos dentre uma série de alternativas
- O caso cuja constante for igual ao valor da expressão será selecionado para execução
- Atenção!
 - Os comandos associados a este caso e todos os comandos seguintes serão executados em sequência até o final do comando switch
 - Para evitar a execução de todos os comandos seguintes, usa-se o comando break

Comando switch

- Para mês = 2:
 - Com o uso do **break**:
 - dias = 28;
 - Sem o uso do **break**:
 - dias = 28;
 - dias = 30;
 - dias = 0;

```
int dias_do_mes3(int mes) {  
    int dias;  
    switch (mes) {  
        case 1:  
        case 3:  
        case 5:  
        case 7:  
        case 8:  
        case 10:  
        case 12:  
            dias = 1;  
            break;  
        case 2:  
            dias = 28;  
            break;  
        case 4:  
        case 6:  
        case 9:  
        case 11:  
            dias = 30;  
            break;  
        default:  
            dias = 0;  
    }  
  
    return dias;  
}
```

A importância dos recuos

Indentação do código

- Programas mais complexos são mais difíceis de ler e compreender
- Uma forma de melhorar a legibilidade do programa é usar **recuos**
- Os recuos devem ser usados sempre após o símbolo {, sendo as instruções recuadas à direita
- O símbolo } deve estar alinhado ao abre-chaves correspondente

A importância dos recuos

Indentação do código

■ Exemplo:

```
if (nota >= 6)
    if (nota_anterior < nota)
        printf("Você está melhorando");
    else
        printf("Voce precisa estudar mais !");
else
    printf("Sem estudo é difícil ser aprovado");
```

A importância dos recuos

Indentação do código

- Recuos não resolvem ambiguidades...

```
if (nota >= 6)
    if (nota_anterior < nota)
        printf("Você está melhorando");
else
    printf("Sem estudo é difícil ser aprovado");
```

- De quem é o **else** acima?
 - O compilador sempre associa um **else** ao if anterior mais próximo que ainda não tiver um else
- Como associar o **else** a instrução `if (nota >= 6)`

A importância dos recuos

Indentação do código

- Exemplo

```
if (nota >= 6){  
    if (nota_anterior < nota)  
        printf("Você está melhorando");  
}  
else  
    printf("Sem estudo é difícil ser aprovado");
```

- Neste caso, as chaves, em vez de opcionais, serão obrigatórias, pois apenas os recuos não resolvem!

Problema 4

- Dado o valor da variável N , determine a soma dos números inteiros de 1 a N
 - Deseja-se calcular o valor de: $1 + 2 + 3 + \dots + N$
- **Observação:**
 - Não sabemos, *a priori*, quantos termos serão somados, pois o valor de N é estabelecido dinamicamente...
- **Como fazer?**
 - Para se calcular esta soma, utiliza-se o comando **while**

O comando **while** permite que um conjunto de instruções seja executado tantas vezes quantas forem necessárias, **enquanto** uma condição for verdadeira

Perguntas?

- E-mail:
 - hector@dcc.ufmg.br
- Material da disciplina:
 - <https://pedroolmo.github.io/teaching/pdsI.html>
- Github:
 - <https://github.com/h3ct0r>



Héctor Azpúrua
h3ct0r