



Técnicas de Programação

TP0101

Prof. Giovane Barcelos

giovane_barcelos@uniritter.edu.br

Plano de Ensino

Conteúdo programático

- 1. Introdução à programação em C**
- 2. Desenvolvimento estruturado de programas em C**
- 3. Controle de programa**
- 4. Funções**
- 5. Arrays**
- 6. Ponteiros**
- 7. Caracteres e strings**
- 8. Entrada/Saída formatada**
- 9. Estruturas, uniões, manipulações de bits e enumerações**
- 10. Processamento de arquivos**
- 11. Estruturas de dados**
- 12. O pré-processador**
- 13. Outros tópicos sobre C**

N1

N2

Estrutura do Programa

Como um programa em C é estruturado?

- Obrigatoriamente um programa C inicia sua execução com a função **main()** que deve especificar um retorno: **void** (vazio), **int** (inteiro)
- Comentários são colocados entre **/*** e ***/** ou iniciam por **//** não sendo considerados na compilação
- Cada instrução finaliza com **;** (ponto e vírgula)
- Os blocos de código são separados por **{** e **}**
- Deve ter o comando **return** ao final

```
void main()  
{  
    printf("Exemplo!"); // Imprime Exemplo!  
    return;  
}
```

Estrutura do Programa

Como um programa em C é estruturado?

- Uma função especifica as ações que um programa executa quando roda
- Há funções básicas que estão definidas na biblioteca C. As funções `printf()` e `scanf()` por exemplo, permitem respectivamente escrever na tela e ler os dados a partir do teclado
- O programador também pode definir novas funções em seus programas, como rotinas para cálculos, impressão, etc. Isto será aprendido ao final da disciplina

Estrutura do Programa

Compilação e execução

- Para executar um programa a partir do seu código fonte é necessário compilá-lo, gerando o código executável. O código executável pode ser executado como qualquer outra aplicação

```
# gcc ola.c -o hello  
# ./ola  
Olá, Mundo!
```

- Como utilizaremos a IDE (Ambiente Integrado de Desenvolvimento)
Visual Studio Code não será necessário a compilação e geração de código pela linha de comando

Estrutura do Programa

Erros de Compilação

- O entendimento dos erros de compilação da Linguagem C é essencial para o desenvolvedor

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

```
void main(){  
    printf("Olá, Mundo!\n");
```

```
#gcc ola.c -o ola
```

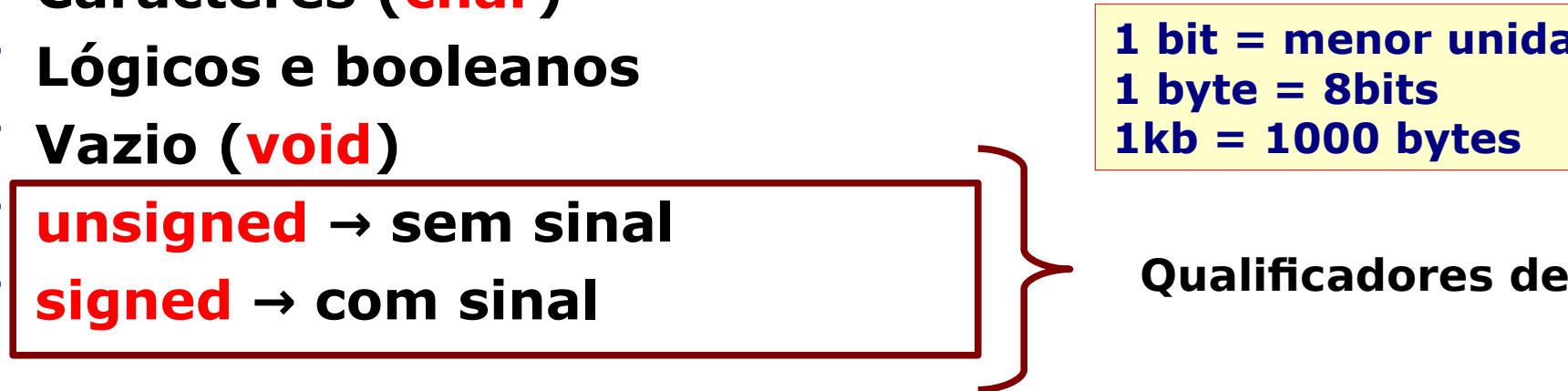
```
ola.c: In function 'main':
```

```
ola.c: aviso: return type of 'main' is not 'int' [-Wmain]
```

```
ola.c:4: error: expected declaration or statement at end of input
```

Variáveis

O que são variáveis?

- As linguagens de programação trabalham com dados
 - Os dados são classificados em tipos
 - Em C temos os seguintes tipos:
 - ✓ Números inteiros (**int**)
 - ✓ Números reais (**float**, **double**)
 - ✓ Caracteres (**char**)
 - ✓ Lógicos e booleanos
 - ✓ Vazio (**void**)
 - ✓ **unsigned** → sem sinal
 - ✓ **signed** → com sinal
- 
- 1 bit = menor unidade (0 ou 1)
1 byte = 8bits
1kb = 1000 bytes
- Qualificadores de Sinal

Variáveis

Quais são os principais tipo de C?

Nome Tipo	Bytes	Limite
int	4	-2147483648 a 2147483647
char	1	-128 a 127
long	4	-2147483648 a 2147483647
float	4	3.4E +/- 38 (7 digitos)
double	8	1.7E +/- 308 (15 digitos)

- Caractere (*char*) pode conter qualquer conjunto de caracteres alfanuméricos (0..9, A..Z, a..z, especiais, #, ?, !, @)
- Char pode conter um único caractere ou uma cadeia de caracteres (*string*)

Variáveis

Qual a classificação dos dados?

➤ **CONSTANTES**

- ✓ **O dado não pode ser alterado no programa**
- ✓ **#define PI = 3.141617; // Nunca será alterado**

➤ **VARIÁVEIS**

- ✓ **O dado pode ser alterado no programa**
- ✓ **Elas são armazenadas temporariamente em memória**
- ✓ **Possuem um nome ou identificador que referenciam a variável**

Variáveis

Como deve ser a nomenclatura das variáveis?

- **Deve começar com um caractere alfabético**
- **Pode conter caracteres alfabéticos e números**
- **Não pode conter caracteres especiais, tais como, espaços, cedilha, sustenido, asterisco, interrogação, exclamação, etc**
- **Não pode ter o mesmo nome de um comando reservado da linguagem C (Ex: if, while, for, etc)**
- **Tamanho máximo de 63 caracteres**
- **Válido: cep, nome, endereco e idade**
- **Inválido: 13Agosto, ?Saldo e (nome)**

Variáveis

Exemplo de declaração de variáveis/constante

```
// Algoritmo 0201.c
// Declaração de uma Constante
# define PI = 3.141617

int main(void)
{
    int idade = 13; // Declaração de variável inteira
    int dia; // Declaração sem valor inicial
    long diaDoAno = 180; // Long
    float salarioMinimo = 622.0; // Float
    double valorProduto = 1308.73; // Double
    char barra = '/'; // Char
    // String. O '*' é um ponteiro de caracteres
    char *nome = "Giovane Barcelos";
    return 0; // Retorna 0, pois main retorna um int
}
```

Saída

Como sair/mostrar dados em C?

- A função **printf** permite mostrar dados na saída ativa, tal como a tela

```
printf("Técnicas de Programação!");
```

- Além de texto puro é possível imprimir conteúdo de variáveis com o **printf**
- Utilizamos o símbolo **%** com o tipo do dado, tal como, **%s** ou **%d**, que no caso é uma **string** (**s**) e um **inteiro** (**d**), para indicar o trecho que deve ser substituído por uma variável

```
curso = "Técnicas de Programação";  
printf("A variável %s contém o valor %d", "curso", curso);
```

Será impresso:

A variável curso contém o valor Técnicas de Programação

Saída

Formatos do printf

- **%d** – Escreve um inteiro na tela sem formatação

```
printf("%d", 13);  
Imprime 13
```

- **%<numero>d** – Escreve um inteiro na tela preenchendo com espaços a esquerda de forma que ele ocupe <numero> caracteres na tela

```
printf("%4d", 13);  
Imprime <espaco><espaco>13
```

<espaco> é um espaço em branco

Saída

Formatos do printf

- **%<numero>d** – Escreve um inteiro na tela preenchendo com espaços a esquerda de forma que ele ocupe <numero> caracteres na tela

```
printf("%4d", 13);  
Imprime <espaco><espaco>13
```

<espaco> é um espaço em branco

- **%0<numero>d** – Escreve um inteiro na tela, preenchendo com zeros a esquerda para que ele ocupe <numero> caracteres na tela

```
printf("%04d", 13);  
Imprime 0013
```

Saída

Formatos do printf

- **%<numero1>.0<numero2>d** – Escreve um inteiro na tela preenchendo com espaços a esquerda de forma que ele ocupe <numero1> caracteres na tela com zeros no comprimento de <numero2>

```
printf("%6.04d", 13);  
Imprime <espaco><espaco>0013
```

<espaco> é um espaço em branco

Saída

Formatos do printf

- A letra **d** pode ser substituída pelas letras **u** e **l** respectivamente quando é necessário escrever variáveis do tipo **unsigned** ou **long**

```
printf ("%d", 4000000000);
```

escreve -294967296 na tela, enquanto que

```
printf ("%u", 4000000000);
```

escreve 4000000000

- **%f** – escreve um ponto flutuante na tela sem formatação

```
printf ("%f", 13.0);
```

imprime 13.000000

Saída

Formatos do printf

- **%e** – escreve um ponto flutuante na tela em notação científica

```
printf ("%e", 13.013131313);  
imprime 1.301313e+01
```

- **%<tamanho>.<decimais>f** – escreve um ponto flutuante na tela com tamanho <tamanho> e <decimais> casas decimais. O ponto utilizado para separar a parte inteira da decimal também conta no tamanho

```
printf ("%6.2f", 13.0);  
Imprime <espaco>13.00
```

Saída

Formatos do printf

- A letra **f** poder ser substituída pelas letras **lf**, para escrever um **double** ao invés de um **float**

```
printf ("%6.2lf", 13.0);  
imprime <espaco>13.00
```

- **%c** – escreve uma letra

```
printf ("%c", 'G');  
imprime G
```

- O **printf ("%c", 71)** também imprime a letra **G**, pois este é o código **ASCII** da letra **G**

- **%s** – escreve uma string

```
printf ("%s", "Técnicas de Programação!");  
imprime Técnicas de Programação!
```

- A função **scanf** permite ler um ou mais dados a partir do teclado
- Parâmetros:
 - ✓ Uma string, indicando os tipos das variáveis que serão lidas e o formato dessa leitura
 - ✓ Uma lista de variáveis
- Esta função espera que o usuário digite um dado e atribui este à variável correspondente

Saída

Como funciona a função scanf?

- O programa abaixo demonstra o funcionamento da função **scanf**

```
// Algoritmo 0202.c
#include <stdio.h>

void main(void)
{
    // Declaração de variável inteira chamada de idade
    int idade;
    // Escreve na tela o que deve ser feito
    printf("Digite a sua idade: ");
    // Faz leitura do tipo inteiro com & de referência
    scanf("%d", &idade);
    // Escreve na tela a idade digitada
    printf("A sua idade é %d!\n", idade);
    return;
}
```

Saída

Outro exemplo

```
// Algoritmo 0203.c
#include <stdio.h>

int main(void)
{
    int num1, num2, num3;

    printf("Digite três números: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    printf("Os valores digitados foram \
        %d %d %d\n", num1, num2, num3);
    return 0;
}
```

O que é o operador & (address-of) em C?

- Toda variável tem um endereço associado a ela
- Esse endereço é o local onde essa variável é armazenada no sistema
- O operador **&** retorna o endereço de uma determinada variável

```
printf ("%d", &valor);
```

Imprime o endereço da variável valor

- É necessário usar o operador **&** no comando **scanf**, pois esse indica que o valor digitado deve ser colocado no endereço da variável
- Esquecer de colocar o **&** é um erro muito comum que pode ocasionar erros de execução

Saída

Exemplo operador &

- O programa abaixo imprime o valor e o endereço da variável

```
// Algoritmo 0204.c
#include <stdio.h>

int main(void)
{
    int valor = 13;
    printf("Valor %d, endereço 0x%x\n", valor, &valor);
    return 0;
}
```

- Os formatos de leitura são muito semelhantes aos formatos de escrita utilizados pelo printf
- A tabela a seguir mostra alguns formatos possíveis de leitura

Código	Função de Leitura
%c	Único caractere
%s	Série de caracteres
%d	Decimal
%u	Decimal sem sinal
%l	Inteiro longo
%f	Ponto flutuante
%lf	Double

Expressões Aritméticas

O que são expressões aritméticas?

- É um conjunto de operações aritméticas, lógicas ou relacionais utilizadas para fazer “cálculos” sobre os valores das variáveis
- As constantes, variáveis e endereços de variáveis também são expressões
- Exemplo:

A + B

Calcula a soma de A e B

Expressões Aritméticas

E como são as atribuições das expressões aritméticas?

- Atribuir em expressões aritméticas significa calcular o valor de uma expressão e copiar o valor resultante para uma determinada variável
- O operador de atribuição é o sinal de igual (=)

Á **esquerda** do operador de atribuição deve existir somente o **nome** da **variável**

=

Á **direita**, deve haver uma **expressão** cujo valor será calculado e armazenado na variável

Expressões Aritméticas

Quais são os tipos mais comuns?

- **<expressao> + <expressao>: calcula a soma de duas expressões.**
Ex: a = a + b;
- **<expressao> - <expressao>: calcula a subtração de duas expressões.**
Ex: a = a - b;
- **<expressao> * <expressao>: calcula o produto de duas expressões.**
Ex: a = a * b;
- **<expressao> / <expressao>: calcula o quociente de duas expressões.**
Ex: a = a / b;
- **<expressao> % <expressao>: calcula o resto da divisão (inteira) de 2 expressões.**
Ex: a = a % b;
- **- <expressao>: inverte o sinal da expressão.**
Ex: a = -b

Expressões Aritméticas

Expressões

- **As expressões aritméticas (e todas as expressões) operam sobre outras expressões**
- **É possível compor expressões complexas como por exemplo: $a = b + 2 + c$**

Qual o valor da expressão $5 + 10 \% 3$?

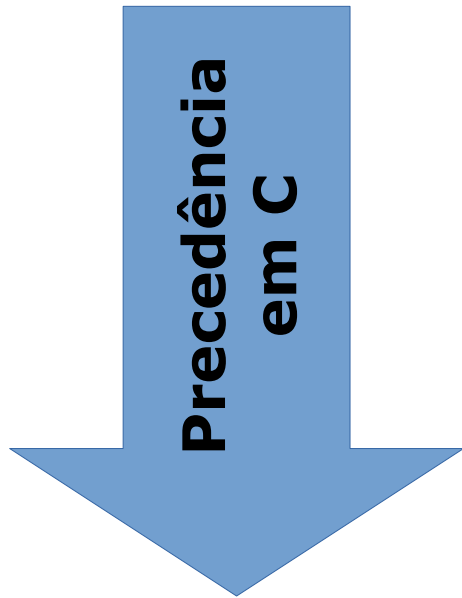
E da expressão $5 * 10 \% 3$?

Para responder estas questões precisamos entender as precedências das expressões

Expressões Aritméticas

O que são precedências?

- Precedência é a ordem na qual os operadores são calculados quando o programa for executado. Em C, os operadores são calculados na seguinte ordem:



- ✓ Parênteses mais internos
- ✓ Funções matemáticas
- ✓ Incremento e decremento
- ✓ Potenciação ou raiz quadrada
- ✓ Multiplicação, divisão, resto ou inteiro
- ✓ Adição ou subtração

- Tu podes utilizar quantos parênteses desejar dentro de uma expressão desde que utilize o mesmo número de parênteses para abrir e fechar expressões

Expressões Aritméticas

Operadores de incremento (++) e decremento(--)

- **Operadores de incremento e decremento tem duas funções: servem como uma expressão e incrementam ou decrementam o valor da variável ao qual estão associados em uma unidade**
- **Exemplo: `dia++;` // incrementa a variável dia em uma unidade**
- **Dependendo da posição do operador de incremento e decremento, uma função é executada antes da outra**

Expressões Aritméticas

Como funciona o operador a esquerda da variável?

- Primeiro a variável é incrementada/decrementada, depois a expressão retorna o valor da expressão

```
// Algoritmo 0205.c
#include <stdio.h>

int main(void)
{
    int numero = 13;
    // Imprime 14, pois primeiro incrementa
    printf("%d", ++numero);
    return 0;
}
```

Expressões Aritméticas

Como funciona o operador a direita da variável?

- Primeiro a expressão retorna o valor da variável e depois a variável é incrementada

```
// Algoritmo 0206.c
#include <stdio.h>

int main(void)
{
    int numero = 13;
    // Imprime 13, pois primeiro retorna o valor
    printf("%d", numero++);
    return 0;
}
```


Expressões Aritméticas

E as atribuições simplificadas

- Em C pode uma implementar atribuições simplificadas
- Uma expressão desta forma: $a = a + b;$
- Pode ser simplificada para: $a += b;$
- Atribuições simplificadas:

Comando	Exemplo	Corresponde a
$+=$	$a += b;$	$a = a + b;$
$-=$	$a -= b;$	$a = a - b;$
$*=$	$a *= b;$	$a = a * b;$
$/=$	$a /= b;$	$a = a / b;$
$\%=$	$a \% = b;$	$a = a \% b;$

Conversão de Tipos

Como converter os tipos?

- É possível converter alguns tipos entre si
- Existem duas formas de fazê-lo: implícita e explícita
- Implícita:
 - ✓ Capacidade (tamanho) do destino deve ser maior que a origem. Ex: `int a; short b; a = b;`
 - ✓ Operações entre `int` e `float` sempre convertem para `float`
- Explícita:
 - ✓ Aplicável a variáveis e expressões. Ex: `a = (int) ((float) b / (float) c);`
 - ✓ Não modifica o tipo "real" da variável, só o valor de uma expressão. Ex: `int a; (float) a = 1.0;` ← Errado

Conversão de Tipos

Uso da conversão de tipos

- A operação de divisão (/) possui dois modos de operação de acordo com os seus argumentos: inteiro ou ponto flutuante
 - ✓ Se os dois argumentos forem inteiros, acontece a divisão inteira. A expressão $10/3$ tem como resultado 3
 - ✓ Se um dos dois argumentos for ponto flutuante, acontece a divisão de ponto flutuante. A expressão $1.5 / 3$ tem como resultado 0.5
- Quando se deseja obter o valor de ponto flutuante de uma divisão (não-exata) de dois inteiros, basta converter um deles para ponto flutuante. A expressão $10 / (\text{float}) 3$ tem como resultado 3.33333333

Expressões

Que tipo de expressões existem?

- **Como estudado anteriormente constantes, variáveis, endereços de variáveis e operações aritméticas são expressões**
- **Mas existem também expressões relacionais e lógicas**

Expressões

O que são expressões relacionais?

- Expressões relacionais são aquelas que realizam uma comparação entre duas expressões que retornam
 - ✓ **Zero (0)**, se o resultado é falso
 - ✓ **Um (1)**, ou qualquer outro número diferente de zero, se o resultado é verdadeiro

Expressões

Valores possíveis

- Numa expressão relacional os valores possíveis são:

S1M **NÃ0**

Expressões

Quais são os operadores relacionais?

Operador	Função
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
==	Igual
!=	Diferente

➤ Exemplos:

$1 + 2 > 3 \rightarrow$ Falso – NÃ**0** é verdade

$3 = 3 \rightarrow$ Verdadeiro – S**1**M, esta correto

Expressões

Expressões Lógicas

- As expressões lógicas atuam sobre expressões e também resultam em valores lógicos (verdadeiro ou falso)
- São utilizados para produzir resultados com base em condições

Operador	Função
&&	Multiplicação Lógica – Resulta VERDADEIRO se ambas as partes forem verdadeiras.
	Adição Lógica – Resulta VERDADEIRO se uma das partes é verdadeira.
!	Negação Lógica – Nega uma afirmação, invertendo o seu valor lógico: se for VERDADEIRO torna-se FALSO, se for FALSO torna-se VERDADEIRO.

Expressões

Operadores Lógicos – Tabela Verdade

- A **Tabela Verdade** é utilizada para construir os resultados com base nos operadores lógicos
- Ela mostra os resultados das aplicações dos operadores lógicos conforme os valores dos operadores envolvidos
- Considere o seguinte:
 - Para efetuar a matrícula é preciso trazer o comprovante de conclusão do segundo grau **E (&&)** passar no vestibular
 - Para fazer cadastro é preciso do número do CPF **OU (||)** RG

Expressões

Operadores Lógicos – Tabela Verdade

Multiplicação	A	B	A && B	
	VERDADEIRO	VERDADEIRO	VERDADEIRO	
	VERDADEIRO	FALSO	FALSO	
	FALSO	VERDADEIRO	FALSO	
	FALSO	FALSO	FALSO	
Adição	A	B	A B	
	VERDADEIRO	VERDADEIRO	VERDADEIRO	
	VERDADEIRO	FALSO	VERDADEIRO	
	FALSO	VERDADEIRO	VERDADEIRO	
	FALSO	FALSO	FALSO	
Negação	A	B	! A	! B
	VERDADEIRO	VERDADEIRO	FALSO	FALSO
	VERDADEIRO	FALSO	FALSO	VERDADEIRO
	FALSO	VERDADEIRO	VERDADEIRO	FALSO
	FALSO	FALSO	VERDADEIRO	VERDADEIRO

Expressões

Operadores Lógicos – Tabela Verdade

➤ 0: Falso ; \geq 1: Verdadeiro

A	B	Multiplicação	Adição	Negação	
		A && B	A B	! A	! B
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	0
0	0	0	0	1	1

Expressões

Tabela Verdade – Exemplo de Uso

▶ **Surfo ou Estudo. Fumo ou não Surfo. Velejo ou não Estudo. Ora, não Velejo. Assim,**

a) Estudo ou Fumo.

b) Não Fumo e Surfo.

c) Não Velejo e não Fumo.

d) Estudo e não Fumo.

e) Fumo e Surfo.

Comandos Condicionais

O que são comandos condicionais?

- Um comando condicional é aquele que permite decidir se um determinado bloco de comandos deve ou não ser executado, a partir do resultado de uma expressão relacional ou lógica



Comandos Condicionais

Comando condicional if

- O principal comando condicional da linguagem C é o **if**, cuja sintaxe é:

```
if (expressao lógica)  
    comando; ou
```

```
if (expressao lógica)  
{  
    comandos  
}
```

- Os comandos são **executados somente** se a expressão lógica for **verdadeira**

Comandos Condicionais

Bloco de Comandos

- É um conjunto de instruções agrupadas
- Limitada pelos caracteres **{** e **}**
- Declaração de variáveis “locais”:
 - ✓ Devem ser sempre declaradas antes de qualquer comando
 - ✓ São válidas somente dentro do bloco

```
int main(void)
{                ← Início do bloco de comandos
    int valor;
    valor = 1;
    return 0;
}                ← Fim do bloco de comandos
```

Comandos Condicionais

Seleção Simples

- A seleção **SIMPLES** TESTA uma certa **CONDIÇÃO**: EXECUTA um **BLOCO** de código SE a **CONDIÇÃO** for **SATISFEITA** ou não faz nada, indo para a próxima depois do bloco

```
// Algoritmo 0301.c
```

```
int main(void)
```

```
{
```

```
    int numero;
```

```
    scanf("%d", &numero);
```

```
    if (numero % 2)
```

Decisão

```
    {
```

```
        printf("O número %d é ímpar!", numero);
```

```
    }
```

```
    return 0;
```

```
}
```

Se condição verdadeira executa o bloco

Comandos Condicionais

Seleção Composta

- Usada quando se quer AVALIAR uma CONDIÇÃO e executar UM BLOCO de código quando VERDADEIRA ou outro BLOCO quando FALSA.

// Algoritmo 0302.c

```
int main(void){
```

```
    int numero;
```

```
    scanf("%d", &numero);
```

```
    if (numero % 2){
```

```
        printf("O número %d é ímpar!", numero);
```

```
    } else {
```

```
        printf("O número %d é par!", numero);
```

```
    }
```

```
    return 0;
```

```
}
```

Decisão

Bloco Verdadeiro

Bloco Falso

Comandos Condicionais

Seleção Encadeada

- Utilizada quando se quer avaliar um grande número de possibilidades

```
// Algoritmo 0303.c
int main(void)
{
    int idade;
    scanf("%d", &idade);
    if (idade >= 18 && idade < 70){
        printf("Obrigatório votar nas eleições!");
    } else {
        if ((idade >= 16 && idade < 18) || idade >= 70) {
            printf("Voto é facultativo nas eleições!");
        } else {
            printf("Não pode votar nas eleições!");
        }
    }
}
```

Comandos Condicionais

Seleção Encadeada

- Utilizada quando se quer avaliar um grande número de possibilidades

```
// Algoritmo 0401.c
int main(void)
{
    int idade;
    scanf("%d", &idade);
    if (idade >= 18 && idade < 70){
        printf("Obrigatório votar nas eleições!");
    } else {
        if ((idade >= 16 && idade < 18) || idade >= 70) {
            printf("Voto é facultativo nas eleições!");
        } else {
            printf("Não pode votar nas eleições!");
        }
    }
}
```

Comandos Condicionais

Múltipla Escolha

- Também utilizada quando se quer avaliar um grande número de possibilidades
- O comando **switch** permite avaliar uma expressão onde uma variável inteira ou caractere deve fazer diferentes operações dependendo exclusivamente de seu valor

Sintaxe

```
switch (variável inteira){  
    case valor: comandos  
    break;  
    case valor: comandos  
    break;  
}
```

Como funciona o comando switch?

- Os comandos começam a ser executados a partir do ponto onde o valor da variável corresponde ao valor antes dos dois pontos (:)
- Executa todos os comandos até que encontre um comando break ou que chegue ao final do bloco de comandos do *switch*
- Pode-se utilizar, ao invés de um valor, o valor *default*. A execução dos comandos inicia no comando default se nenhum outro valor for correspondente ao valor da variável

Comandos Condicionais

Exemplo de Múltipla Escolha

```
// Algoritmo 0402.c
#include <stdio.h>
int main(void){
    int codigo;
    scanf("%d", &codigo);
    switch (codigo) {
        case 1:
            printf("Código 1!");
            break;
        case 2:
            printf("Código 2!");
            break;
        default:
            printf("Outro código!");
            break; }
    return 0;
}
```

Debug

Para que serve e como utilizar o Debug?

- O Debug serve para investigarmos erros de programação e de lógica nas aplicações
- Esta ferramenta permite executar aplicações passo a passo, linha por linha, e visualizar todos os aspectos que envolvem chamadas de funções e variáveis
- Vamos testar e aplicarmos o debug numa aplicação

Repetição

O que é uma estrutura de repetição?

- **Estrutura de repetição é uma estrutura de desvio de fluxo de controle que repete um conjunto de instruções computacionais dependendo de uma condição ser verdadeira ou falsa**
- **Ou seja, repete um bloco de instruções dependentes de uma condição**
- **Repete um bloco de código um número determinado ou indeterminado de vezes**
- **A estrutura de repetição é útil para reduzir e viabilizar blocos de código. Imagine que fosse necessário escrever um programa que mostrasse os 1000 primeiros números inteiros positivos começando em um. Como isto seria possível?**

Repetição

while (condição) { comandos }

➤ A estrutura **while** executa um **bloco de comandos** enquanto uma dada **condição** for **verdadeira**

➤ Estrutura:

while (condicao) comando;

while (condicao) { comandos }

Repetição

Exemplo while

- Mostrar os 1000 primeiros números inteiros positivos começando em um

```
// Algoritmo 0501.c
```

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

```
int main(void)
```

```
{
```

```
    int num = 1;
```

```
    while (num < 1001){  
        printf("%d\n", num);
```

```
        num++;
```

```
    }
```

```
    return 0;
```

```
}
```

Condição de Repetição

Passo ou Incremento da
Variável de Controle de
Repetição

Repetição

Exemplo while

- Mostrar os N primeiros números inteiros positivos começando em um

```
// Algoritmo 0502.c
#include <stdio.h>

int main(void)
{
    int num = 1, nPrim;
    scanf("%d", &nPrim);
    while (num <= nPrim){
        printf("%d\n", num);
        num++;
    }
    return 0;
}
```

Repetição

while

- O que acontece se a condição for falsa na primeira vez?

```
while (a != a) a = a + 1;
```

- O que acontece se a condição for sempre verdadeira?

```
while (a == a) a = a + 1;
```

Repetição

Determinada

➤ Loop (repetição) determinado:

```
scanf("%d", &preco);  
while (i<=n) {  
    total = total + preco;  
    i++;  
    scanf("%d", &preco);  
}
```

➤ Loop (repetição) indeterminado:

```
scanf("%d", &preco);  
while (preco > 0) {  
    total = total + preco;  
    scanf("%d", &preco);  
}
```

Repetição

do { comandos } while (condicao);

- Da mesma forma que o **while**, A estrutura **do while** executa um **bloco de comandos** enquanto uma dada **condição** for **verdadeira**
- A diferença em relação ao **while** é que ela **SEMPRE** entra no bloco de comandos na primeira vez, visto que, a condição é verificada ao final do bloco
- Estrutura:
do comando; while (condição);
do { comandos } while (condição) ;

Repetição

Exemplo do while

- Algoritmo que percorre os números de 1 a 10 e classifica estes em pares e impares.

```
// Algoritmo 0503.c
#include <stdio.h>

int main(void){
    int numero = 1;
    do {
        if ((numero % 2) == 0){
            printf("O número %d é par!\n", numero);
        } else {
            printf("O número %d é impar!\n", numero);
        }
        numero++;
    } while ( numero < 11);

    return 0;}

```

Repetição

Problemas

- Os principais problemas encontrados com o **while** e **do while** são:
 - ✓ Onde são inicializadas as variáveis usadas na condição da repetição?
 - ✓ A variável de controle da condição, também chamada de passo, pode estar em qualquer ponto da repetição

Repetição

Problemas - Exemplo

```
i = 0;
```

```
/* várias linhas de código */
```

```
while (i < 10) {
```

```
    j = j * 2;
```

```
    l = j - i;
```

```
    i++;
```

```
    k = i + j;
```

```
}
```

Inicialização de i

Condição da Repetição

Passo ou Incremento da Variável de Controle

Repetição

for (início ; condição ; passo) { comandos ;}

- O comando **for** resolve os problemas apresentados anteriormente pelo **while** e **do while**
- Estrutura:
for (início; condição; passo) comando;
for (início; condição; passo) { comandos };
- Início: Uma ou mais atribuições e/ou declarações de variáveis, separadas por “,”
- Condição: Idêntico ao **while** e **do while**
- Passo: Um ou mais comandos, separados por “,”

Repetição

Exemplo for

- Mostrar os 1000 primeiros números inteiros positivos começando em um

```
// Algoritmo 0504.c
#include <stdio.h>
int main(void){
    for ( int num = 1; num < 1001; num++)
        printf("%d\n", num);
    return 0;
}
```

- ✓ Para funcionar no Code::blocks colocar **-std=c99** em **Settings** → **Compiler and debugger** → **Compiler settings** → **Other options**

Repetição

Exemplo for

- Mostrar os N primeiros números inteiros positivos começando em um

```
// Algoritmo 0505.c
#include <stdio.h>

int main(void)
{
    int nPrim;
    scanf("%d", &nPrim);
    for ( int num = 1; num <= nPrim; num++)
        printf("%d\n", num);
    return 0;
}
```

Repetição

Quando usar o **for**, **while** e **do while** ?

- Do ponto de vista de implementação qualquer um deles é intercambiável
- Entretanto, para clareza de código é sugerido:
 - ✓ Se o código é uma repetição determinada utilize sempre o **for**
 - ✓ Se a repetição é indeterminada e o bloco de código só pode ser iniciado se a condição for verdadeira, dê preferência ao **while**
 - ✓ Se a repetição é indeterminada e o bloco deve ser iniciado sempre na primeira vez independente da condição ser verdadeira, de preferência ao **do while**
 - ✓ Esta em dúvida de qual utilizar, implemente com o **for**

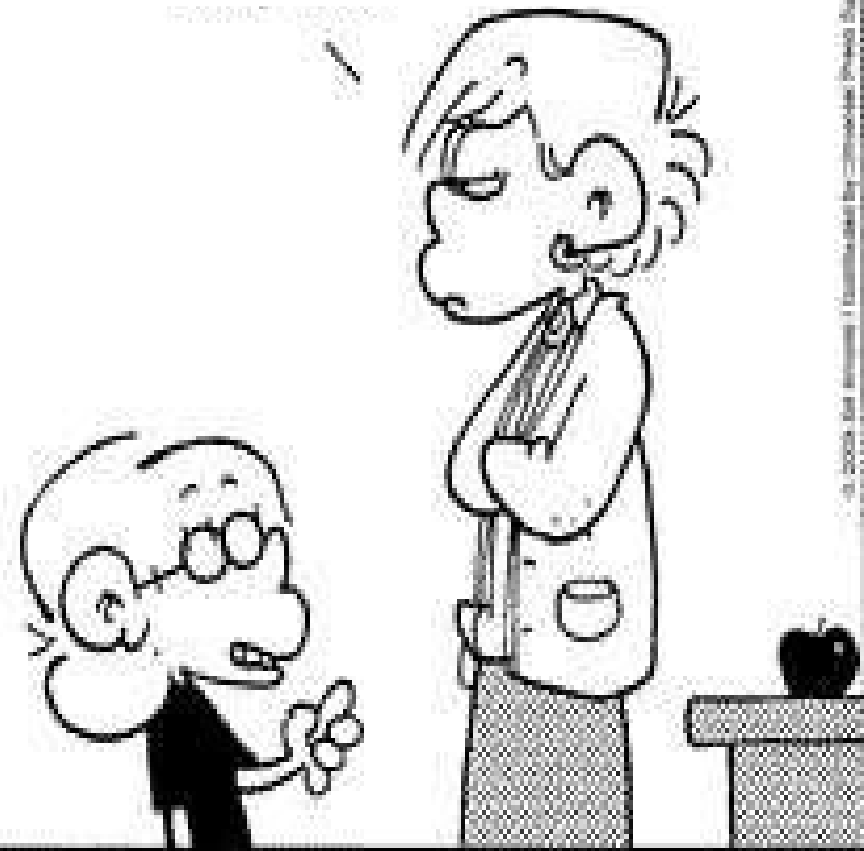
Repetição

Uso Prático

**Castigo: Escreva 500 vezes no quadro a frase
"Não vou falar mais durante a aula!"**

```
#include <stdio.h>
int main(void){
    int conta;
    for (conta = 1; conta <= 501; conta++){
        printf("Não vou falar mais durante a aula!\n");
    }
    return;
}
```

Boa Tentativa!



“ Uma longa viagem começa com um único passo. ”



Lao Tsé