

Linguagem C

Tipos de dados e operadores



Tipos de dados


- Os 4 tipos de dados básicos

Tipo	Intervalo	Bits
char	-127 a 127	8
int	-32.767 a 32.767	16
float	6 casas decimais	32
double	10 casas decimais	64

- Como escolher o tipo de dado para cada caso

Modificador short/long

- Permite aumentar o tamanho do tipo de dado
- Válido somente para int e double



Modificador short/long

Tipo	Intervalo	Bits
int	-32.767 a 32.767	16
short int		
long int	-2.147.483.647 a 2.147.483.647	32
double	10 casas decimais	64
short double		
long double	10 casas decimais	80

Modificador signed/unsigned

- Gerenciar somente números positivos
- float e double não utilizam esta propriedade

Modificador signed/unsigned

Tipo	Intervalo	Bits
char	-127 a 127	8
signed char		
unsigned char	0 a 255	8
int	-32.767 a 32.767	16
signed int		
short int		
signed short int		
unsigned int	0 a 65.535	16
long int	-2.147.483.647 a 2.147.483.647	32
signed long int		
unsigned long int	0 a 4.294.967.295	32

Tipos de dados da linguagem C



Tipo	Intervalo	Bits
char	-127 a 127	8
signed char		
unsigned char	0 a 255	8
int	-32.767 a 32.767	16
signed int		
short int		
signed short int		
unsigned int	0 a 65.535	16
long int	-2.147.483.647 a 2.147.483.647	32
signed long int		
unsigned long int	0 a 4.294.967.295	32
float	6 casas decimais	32
double	10 casas decimais	64
long double	10 casas decimais	80

Variáveis



- Espaços de memória alocados para armazenar valores
- Nomenclatura
 - Não utilizar caracteres especiais
 - Não utilizar espaço em branco ou pontuações
 - Não começar com números
 - Case sensitive
- Sintaxe: `tipo nome`

Exemplos de variáveis



- Exemplos de criações incorretas de variáveis

```
int carro grande; // Espaço em branco não deve ser utilizado
char 15andar;    // Não deve iniciar com caractere numérico
double numeração; // Acentos e cedilha não devem ser utilizados
float double var; // Cada variável pode ser somente de um tipo de cada vez
```

- Exemplos de criações correta de variáveis

```
char minhaVariavel; // Tudo junto, sem acento
int idade_da_pessoa; // underline é permitido
int x, y, z; // Declarando 3 variáveis na mesma linha, de mesmo tipo
double peso1; // Utilizando número no nome da variável
```

Variáveis locais



- Escopo local (bloco de código)
- Automaticamente excluídas no final do bloco
- Devem ser declaradas no início de cada bloco

```
#include <stdio.h>

int main(void)
{
    char x;           // Criando uma variável do tipo char
    double carro1, carro2; // Criando duas variáveis na mesma linha
    int a = 15;        // Criando e já inicializando uma variável

    // Utilização das variáveis
}
```

Variáveis globais



- Escopo global
- Ocupam memória durante toda a execução
- Devem ser declaradas fora dos blocos de código, no início do arquivo

```
#include <stdio.h>

int idade;
char nome;

int main(void)
{
    // Utilização das variáveis
}

void outroBloco(void)
{
    // Utilização das variáveis
}
```

Atribuições



- Operador igual (=)
- Vincula o valor apresentado em seu lado direito na variável existente do lado esquerdo

```
int x = 12;           // Atribui o valor 12 à variável x
double y = 75.06;    // Atribui o valor 75.06 à variável y
```

- Atribuição múltipla

```
int x, y, z;
x = y = z = 3;        // Atribui o valor 3 para z, para y e para x
```

Hexadecimais e octais



- Hexadecimais

```
int minhaVarHexa = 0x32; // Número 50 em decimal
int x = 0x145; // Número 325 em decimal
```

- Octais

```
int minhaVarOcta = 062; // Número 50 em decimal
int y = 0505; // Número 325 em decimal
```

Modificadores de armazenamento



- extern

- Resolve o conflito de variáveis e constantes globais entre diferentes programas em C
- Defina no primeiro programa sem a instrução extern, e com ela nos demais programas

- static

- Variáveis que mantêm seus valores entre diferentes chamadas de uma mesma função
- Variáveis globais que compartilham seu valor dentro do mesmo arquivo (código-fonte)

Modificadores de armazenamento



- register

- Armazena a variável no mecanismo de acesso mais rápido disponível

```
extern int frota;
static int chamadas = 30;
register int contador;
double dolar = 2.10;
```

Constantes



- **const**
 - Valores constantes que não podem ser alterados durante a execução do programa
 - Não pode ser inicializadas em separado
- **volatile**
 - Contantes que podem sofrer alterações por vias externas ao seu programa

```
const double PI = 3.14;           // Constante do tipo int
volatile int processo = 50;       // Constante VOLATILE
const int mesesQueTemNoAno;       // Constante sem ser inicializada
mesesQueTemNoAno = 12;           // Erro ao inicializá-la
```

Enumerações



- Grupo de valores que uma variável pode ter
- Limitar o valor de uma variável em um subconjunto conhecido de valores
- Sintaxe: enum **nome** {valores}

```
enum estaçõesDoAno {verao, outono, inverno, primavera};
enum estaçõesDoAno x;
x = verao;
x = outono;
```

Operações matemáticas



Para os exemplos apresentados, assuma:

```
double x;           // Variável "x" declarada como do tipo de dado double
= Atribui o valor da sua direita à variável a sua esquerda
x = 3;              // Resultado: x armazena o valor 3
+ Soma dois valores numéricos
x = 3 + 5;          // Resultado: x armazena o valor 8
- Subtrai dois valores numéricos
x = 3 - 5;          // Resultado: x armazena o valor -2
* Multiplica dois valores numéricos
x = 2 * 6;          // Resultado: x armazena o valor 12
/ Divide dois valores numéricos
x = 18 / 3;         // Resultado: x armazena o valor 6
% Obtém o resto da divisão entre dois valores numéricos
x = 19 % 3;         // Resultado: x armazena o valor 1
```

Operações matemáticas



++ Incrementa em 1 o valor da variável acoplada

```
x = 3;
x++; // Resultado: x neste momento armazena o valor 4
```

-- Decrementa em 1 o valor da variável acoplada

```
x = 3;
x--; // Resultado: x neste momento armazena o valor 2
```

• É possível utilizar estes operadores em expressões e outros comandos

```
x = 3;
y = 2 + x++; // Resultado: y = 5 e x = 4
```

• Diferença entre ++x e x++

```
x = 3;
y = 2 + ++x; // Resultado: y = 6 e x = 4
```

Operações matemáticas



Assumir x = 5 para o início de cada exemplo apresentado neste slide

+= Soma à variável à sua esquerda o valor apresentado em sua direita

```
x += 3; // Resultado: x armazena o valor 8
```

-= Subtrai da variável à sua esquerda o valor apresentado em sua direita

```
x -= 3; // Resultado: x armazena o valor 2
```

*= Multiplica a variável à sua esquerda pelo valor apresentado em sua direita

```
x *= 3; // Resultado: x armazena o valor 15
```

/= Divide a variável à sua esquerda o valor apresentado em sua direita

```
x /= 3; // Resultado: x armazena o valor 1.6666666666667
```

%= Atribui à variável da esquerda o resto de sua divisão pelo valor apresentado em sua direita

```
x %= 3; // Resultado: x armazena o valor 2
```

Conversões de tipos de dados



- Operações matemáticas entre diferentes tipos de dados podem ocasionar imprecisão
- Conversão automática
- Conversão implícita (cast)

```
int x = 10;
int y = 3;

int resultadoInt = x / y;
printf("%d\n", resultadoInt); // 3
printf("%f\n", resultadoInt); // 0.000000

double resultadoDouble = x / y;
printf("%f\n", resultadoDouble); // 3.000000

double resultadoDoubleCast = (double) x / y;
printf("%f\n", resultadoDoubleCast); // 3.333333
```

```
double x = 10;
int y = 3;

int resultadoInt = x / y;
printf("%d\n", resultadoInt); // 3
printf("%f\n", resultadoInt); // 0.000000

double resultadoDouble = x / y;
printf("%f\n", resultadoDouble); // 3.333333

double resultadoDoubleCast = (double) x / y;
printf("%f\n", resultadoDoubleCast); // 3.333333
```

Operadores bit a bit



- A linguagem C suporta algumas operações de baixo nível, como as operações bit a bit
- Representados pelos tipos int e char

Operador	Operação	
&	AND ("e" lógico)	int x = 6; // 0110 int y = 3; // 0011
	OR ("ou" lógico)	int z = x & y; // 0110 AND 0011 = 0010 printf("%d", z); // 0010 é o decimal 2
^	XOR ("ou" exclusivo)	int w = x ^ y; // 0110 XOR 0011 = 0101 printf("%d", w); // 0101 é o decimal 5
~	Complemento de um	
>>	Deslocamento para a direita	int q = x >> 2; // 0110 >> 2 = 0001 printf("%d", q); // 0001 é o decimal 1
<<	Deslocamento para a esquerda	

Diretivas (#)



- Instruções pré-processadas pelo compilador
- Tempo de compilação vs. Uso de variáveis
- Cada uso de uma diretiva deve ser realizado em uma linha própria
 - #define, #undef
 - #if, #else, #elif, #endif
 - #ifdef, #ifndef
 - #line, #error
 - #include

#define, #undef



- #define
 - Permite atrelar um valor por meio de um identificador


```
#define PI 3.14
```
 - Pode ser utilizado para definir uma breve função


```
#define PAR_OU_IMPAR(numero) numero%2==0 ? 0 : 1
```
- #undef
 - Remove a definição de uma chave


```
#undef PI
```


#if, #else, #elif, #endif



- **#if**
 - Analisa uma condição e inicia o bloco afirmativo
- **#else**
 - Encerra o bloco anterior e inicia o negativo
- **#elif**
 - Encerra o bloco anterior e abre outro condicional
- **#endif**
 - Encerra o comando como um todo

```
#if PI > 5
// Código caso afirmativo
#endif

#if PI > 5
// Código caso afirmativo
#else
// Código caso negativo
#endif

#if PI > 5
// Código caso afirmativo
#elif PI < 2
// Código caso afirmativo (condição elif)
#else
// Código caso negativo (condição elif)
#endif
```

#ifdef, #ifndef



- **#ifdef**
 - Verifica se uma chave foi definida
- **#ifndef**
 - Verifica se uma chave não foi definida
- Ambas permitem o uso de **#else**

```
#ifdef PI
// Código caso afirmativo
#endif
```

```
#ifndef PI
// Código caso afirmativo
#endif
```


#line, #error



- **#line**
 - Gerencia o número da linha
 - Inicialização opcional
 - Identificadores `__LINE__` e `__FILE__`
- **#error**
 - Interrompe a compilação

```
#line 5 // Inicia o contador de linhas
// Linha 5
printf("%d", __LINE__); // Linha 6
printf("%d do arquivo %s", __LINE__, __FILE__); // Linha 7
```

#include



- #include
 - Orienta o compilador para ler outro arquivo
 - Comando de importação de bibliotecas

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