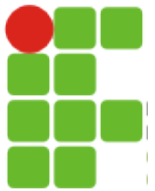
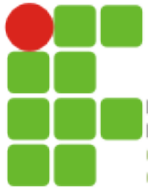
Expressões em C

Prof. Daniel Ferreira

Instituto Federal do Ceará
Campus Maracanaú

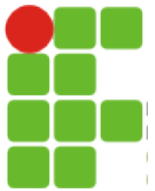


Sumário



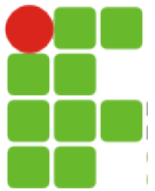
Sumário

Introdução



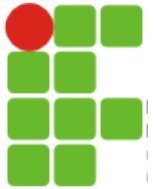
Introdução

- Uma expressão é o elemento mais fundamental da linguagem.
- Expressões são formados por dados e operadores.
- C possui 5 tipos básicos de dados.



Objetivo

Caracterizar expressões e seus componentes, em linguagem C.



Tipos Básicos em C

1. Os tipos básico são os seguintes:
   - caractere (char),
   - inteiro (int),
   - ponto flutuante (float),
   - ponto flutuante com dupla precisão (double),
   - sem valor (void).
2. C ANSI especifica apenas a faixa mínima, não o seu tamanho.
3. sizeof(nome_variavel) retorna o tamanho do tipo da variável.

Tipos Básicos em C

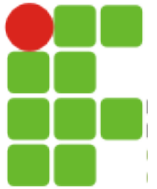
Tipo	Tamanho [1]
char	8 bits
int	32 bits
float	32 bits
double	64 bits

[1] Depende da plataforma subjacente.

Modificando Tipos

As palavras-chaves a seguir, podem modificar os tipos, a saber:

- signed,
- unsigned,
- long,
- short.



Modificando Tipos

Exercício: Complete a tabela com o tamanho dos tipos básicos, bem como os mesmos modificados, conforme seu computador.

Tipo	Tamanho	Faixa mínima
char	8 bits	−127 a 127
unsigned char	8 bits	0 a 255
signed char	8 bits	−127 a 127
int		
unsigned int		
signed int		
short int		
unsigned short int		
signed short int		
long int		
signed long int		
unsigned long int		
float		Quantos dígitos de precisão?
double		Quantos dígitos de precisão?
long double		Quantos dígitos de precisão?

Nomes de Identificadores

Os nomes dos identificadores devem seguir as seguintes regras:

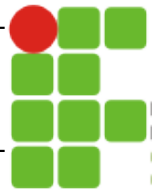
- o primeiro caractere deve ser um caractere ou um sublinhado;
- os caracteres subsequentes devem ser letras, números ou sublinhados.
- não podem ter mesmo nome que as palavras-chave.
- não podem ter mesmo nome de suas funções ou da biblioteca C.

Exemplos 1:

Correto	**Incorreto**
count	1count
test23	hi!there
high_balance	high...balance

Exemplos 2:

Correto
nomes_de_identificadores_execessivamente_longos_sao_incomodos
nomes_de_identificadores_execessivamente_longos_sao_incomodos_para_usar



COUNT, count e Count são identificadores diferentes.

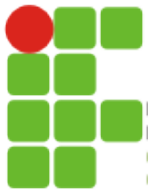
Variáveis

- Variável é uma posição nomeada de memória, que é usada para guardar um valor.
- A forma geral de uma declaração é:
  1. tipo_de_dados lista_de_variaveis;
- Exemplo:
  1. int a, b, c;
  2. unsigned int d;
- Inicialização de variáveis:
  1. int a = 2;
  2. int a = 2, b = 3, c = 4;
- Operador de atribuição (=)[2] é aquela no qual o valor resultante de uma expressão é copiado para a variável.

[2] Não é equivalente a igualdade da matematica.

Exercício

1. Crie uma variável para cada um dos 5 tipos básicos e use sizeof(...) em conjunto com printf(...) para mostrar o tamanho dos tipos básicos em C na arquitetura do seu computador.
2. Declare duas variáveis (**int a = 3, b = 2;**) e troque os valores armazenados nelas, de forma que a variável **a** contenho o valor da variável **b** e vice-versa (Obs: use a operção de atribuição, logicamente).
3. Extrapole o valor limite de variáveis do tipo inteiro. Imprima o resultado e veja o que acontece.



Onde são declaradas?

As variáveis são declaradas em três lugares:

- dentro de funções (variáveis locais);
- na definição das funções (parâmetros formais);
- fora de qualquer função (variáveis globais).

Variáveis Locais

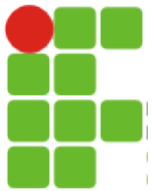
Variáveis locais são declaradas dentro de funções.

Exemplo:

```
int main()
{
    int a;
    int b = 3;
    printf("%d %d",a, b);
}
```

Considerações importantes:

- as variáveis **a** e **b** só existem dentro das chaves;
- elas são criadas a partir de sua declaração [3];
- elas são destruídas na saída do bloco de código.



[3] A maioria dos programadores declara as variáveis locais no começo das funções. Não é obrigatório em alguns compiladores.

Variáveis Locais

Outro exemplo:

```
int func1()
{
    int x;
    x = 2;
}
int func2()
{
    int x;
    x = 3;
}
```

Considerações importantes:

- a variável **x** de **func1()** e **x** de **func2()** são diferentes;
- elas ocupam regiões de memória diferentes;
- só existem dentro de seu bloco de código.

Palavra-chave **auto**: por padrão toda variável não-global é **auto**mática.[4]

[4] Pode-se usar a palavra-chave auto antes de declarar uma variável local.

Variáveis Locais

Outro exemplo:

```
int funcao()
{
    int x, y = 3;
    x = 10;
   {
       int z, y = 2;
       z = 100;
       x = 50;
   }
}
```

Considerações importantes:

- a variável **x** existe dentro de todo o bloco da função, inclusive no bloco de código dentro do bloco de código da função;
- a variável **y** definida no começo da função é diferente da variável **y** dentro do bloco de código interno a função;

Parâmetros Formais

Na função

```
int soma(int n1, int n2)
{
    ...
}
```

podemos afirmar que:

- **n1** e **n2** são parâmetros formais da função **soma()**;
- **n1** e **n2** tem o mesmo significado que qualquer variável local declarada;
- sua declaração ocorre dentro dos parêntesis na definição da função;
- como variáveis locais, são criadas no início da função e destruídas ao final;
- o programador deve ter cuidado para passar valores que correspondam aos tipos definidos;
- **n1** e **n2** são inicializadas com os valores passados quando da chamada a função.

Variáveis Globais

São reconhecidas em todo o programa. E possuem as seguintes características:

- existem durante todo o programa;
- são declaradas fora de qualquer função [5].
- é melhor declará-las no início do programa;
- estão armazenadas em uma região fixa, reservada para este propósito;
- deve-se evitar variáveis globais desnecessárias, pois ocupam espaço durante toda a execução.

[5] Seja antes ou depois das funções que as utilizam

Variáveis Globais

Exemplo:

```
int count;
int main()
{
   count = 100;
   func1();
}
int func1()
{
   int tempo = count;
   func2();
}
int func2()
{
   int count = 3;
}
```

Variáveis Globais

Considerações, sobre o código anterior:

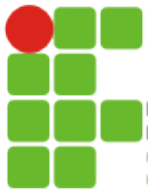
- usa-se **count** nas funções **main()** e **func1()**, verifique que não foram declaradas localmente.
- na função **func2()** há a declaração de **count**, qualquer referência dentro do seu bloco de código se refere a variável local.
- a variável local **count** tem vida útil menor que a variável global **count**.

Escopo: é o denominação dada para a região ou bloco de código no qual a variável existe.

Modificadores de Tipo de Acesso

Modificadores ou Quantificadores: controlam a maneira como as variáveis podem ser modificadas. Os modificaores de tipo de acesso, são:

- const; e
- volatile.



Modificador: const

Características de variáveis do tipo **const**:

- não podem ser modificada por seu programa;
- podem ou não ser alteradas por um processo externo ao programa;
- podem ser usada em outras expressões;
- permitem proteger variáveis de alterações pelo programa;
- muitas funções da biblioteca C padrão usam **const**;
- podem receber um valor inicial;
- uso:
  1. const int c = 10;

Modificador: volatile

Características de variáveis do tipo **volatile**:

- informa que o valor de uma variável pode ser alterado por processo externo ao programa;
- valor é alterado sem sem nenhum comando de atribuição explícito;
- tem seu uso devido otimização de alguns compiladores;
- previne a não reexaminação de valores de variáveis;
- uso:
  1. volatile int v;

O uso de **const e volatile** é possível:

1. const volatile unsigned char *port = 0x30;

Especificadores de Tipo de Classe de Armazenamento

- São usados para informar ao compilador como a variável deve ser armazenada.
- Há quatro tipos a saber:
  1. extern;
  2. static;
  3. register;
  4. auto.
- Tem seu uso da seguinte forma:
  1. especificador_de_tipo tipo nome_variavel;

Especificadores de Tipo: extern

- Existência de módulos em programas C;
- Usar variáveis globais definidas em outros módulos;
- Evitar redeclarar variáveis. [6]
- O problema ocorreria na linkedição dos módulos.
- Solução? extern.

[6] O compilador pode emitir mensagem de erro, ou considerar as duas variáveis como diferentes.

Especificadores de Tipo: extern

Exemplo de uso em arquivos diferentes:

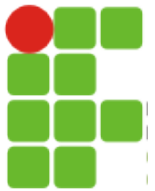
Arquivo 1	**Arquivo 2**
int x,y;	extern int x,y;
char ch;	extern char ch;
int main()	int func22()
{	{
.	x = y/10;
.	return x;
.	}
}	
	int func23()
int func1()	{
{	y = 10;
x = 123;	return y;
return x;	}
}	

Especificadores de Tipo: extern

Exemplo de uso no mesmo arquivo, a saber:

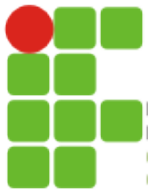
Arquivo 1

```
int first, last;          * declaração global*
int main()
{
    extern int first;     * declaração opcional*
}
```



Especificadores de Tipo: static

- Variáveis **static** são variáveis permanentes;
- São reconhecidas apenas dentro do arquivo ou função de declaração;
- Mantêm seus valores entre chamadas;
- São boas para definição de funções generalizadas e de biblioteca;
- O especificador **static** tem efeitos diferentes em variáveis locais e globais.

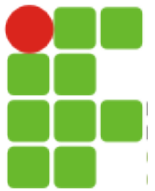


Variáveis Locais static

- Tem armazenamento permanente como as variáveis globais;
- Pode ser reconhecida apenas na função onde foi declarada;
- Logo, retém seu valor entre chamadas;

Exemplo:

```
int series()
{
static int series_num = 100;
series_num = series_num + 23;
return series_num;
}
```



Variáveis Globais static

- Tem armazenamento permanente como as variáveis globais;
- Pode ser reconhecida apenas no arquivo onde foi declarada;
- Variáveis não podem ser referenciadas em outros arquivos.

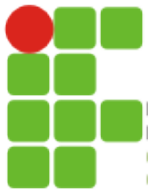
```
static int series_num;
void series_start(int seed);
int series();
int series()
{
series_num = series_num + 23;
return series_num;
}
int series_start(int seed)
{
series_num = seed;
}
```

Especificadores de Tipo: register

- O acesso a uma variável **register** deve ser o mais rápido possível.
- Antes, o especificador **register** era aplicado apenas aos tipos **int** e **char**.
- Antes, a definição C ANSI exigia que os tipos ficassem em registradores.
- Hoje, na prática caracteres e inteiros são colocados nos registrados;
- Enquanto, objetos maiores como matrizes têm tratamento especial.
- O padrão C ANSI permite que seja ignorado o especificador **register**.
- O especificador **register** só pode ser aplicado a variáveis locais e parâmetros formais.
- Variáveis register não tem endereço. Por que?

Especificadores de Tipo: register

```
int pwr(register int m, register int e)
{
register int temp;
temp = 1;
for(; e; e–)
    temp = temp * m;
return temp;
}
```



Constantes

Tipo de Dado	Exemplos de Constantes
char	'a' 'b'
int	1123 21000 −234
long int	35000L −34L
short int	10 −12 90
unsigned int	10000U 987U 40000
float	123.23F 4.34$e-3F$
double	123.23 12312333 −0.9876324
long double	101.2L

- int hex = 0x80 /* 128 em decimal */
- int oct = 012 /* 10 em decimal */
- char str[] = "isso é um teste!";

Operadores

- C é rica em operadores.
- Contêm 4 grupos de operadores: aritméticos, relacionais, lógicos e bit-a-bit.
- Além desses, alguns especiais: operadores de atribuição;
  1. nome_da_variavel = expressão;
- e conversão de tipo [7].

```
1 int x;
2 char ch;
3 float f;
4 void func()
5 { ch = x; /* de int para char */
6 x = f; /* de float para int */
7 f = ch; /* de char para float */
8 f = x; /* de int para float */
9 }
```

[7] C permite conversão automática de tipos.

Operadores Aritméticos

Operador	Ação
-	subtração, menos unário
+	Adição
*	Multiplicação
/	Divisão
%	Módulo da divisão (resto)
- -	Decremento
++	Incremento

Precedência[8]	OPerador
Mais alta	++ - -
	- (menos unário)
	* / %
Mais baixa	+ -

[8] Operadores com mesma precedência são avaliados da esquerda para a direita.

Operadores Relacionais e Lógicos

Operador Relacional	Ação
>	Maior que
>=	Maior que ou igual
<	Menor que
<=	Menor que ou igual
==	Igual
!=	Diferente

Operador Lógico	Ação
&&	AND
||	OR
!	NOT

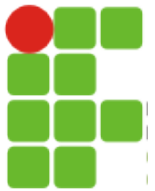
Operadores Relacionais e Lógicos

P	Q	P && Q	P || Q	!P
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Precedência	Operador
Mais alta	!
	> >= < <=
	== !=
	&&
Mais baixa	||

Operadores Bit a Bit

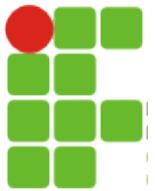
- C foi projetada para substituir assembly.
- Assim, deve suportar as diversas operações a nível de bit.
- Operações tais como: testar, atribuir ou deslocar os bits.
- Operações em bits não podem ser realizadas em float, double, long double, void [9].



[9] Devem ser realizadas em variáveis do tipo char e int ou variantes

Operadores Bit a Bit

Operador Bit-a-Bit	Ação
&	AND
|	OR
∧	XOR
~	Complemento de um
<<	Deslocamento à esquerda
>>	Deslocamento à direita



Operador AND bit-a-bit

- Esse operdaor desliga bits. Como assim?

1 1 0 0 0 0 0 1	ch contém 'A' com a paridade ligada
0 1 1 1 1 1 1 1	127 em binário
& ———————	faz AND bit-a-bit
0 1 0 0 0 0 0 1	ch contém 'A' sem paridade

Operador OR bit-a-bit

- Esse operdaor liga bits. Como assim?

1 0 0 0 0 0 0 0	128 em binário
0 0 0 0 0 0 1 1	3 em binário
| ————————	faz OR bit-a-bit
1 0 0 0 0 0 1 1	resultado

Operador XOR bit-a-bit

- Esse operdaor liga bits se forem diferentes. Como assim?

0 1 1 1 1 1 1 1	128 em binário
0 1 1 1 1 0 0 0	3 em binário
$\wedge$ ———————	faz XOR bit-a-bit
0 0 0 0 0 1 1 1	resultado

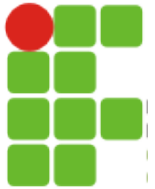
Deslocamento à esquerda e à direita

- Divide ou multiplica por multiplos de 2. Como assim?

unsigned char x;	**x a cada execução**	**valor de x**
$x = 7;$	0 0 0 0 0 1 1 1	7
$x = x << 1;$	0 0 0 0 1 1 1 0	14
$x = x << 3;$	0 1 1 1 0 0 0 0	112
$x = x << 2;$	1 1 0 0 0 0 0 0	192
$x = x >> 1;$	0 1 1 0 0 0 0 0	96
$x = x >> 2;$	0 0 0 1 1 0 0 0	24

Exercício

1. Implemente o OU EXCLUSIVO.



Operador Complemento de Um

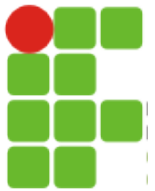
- Esse operador inverte todos os bits (Ou seja é o NOT bit-a-bit).

```
0 0 1 0 1 1 0 0    byte original
1 1 0 1 0 0 1 1    após 1º complemento
0 0 1 0 1 1 0 0    após 2º complemento (igual ao byte original)
```

```
/* Uma função simples e criptografia*/
char encode()
{
return ~ch; /* complemento de um */
}
```

Operador Ternário (?)

- A seguinte expressão: op = Exp1 ? Exp2 : Exp3;
- Equivale a: if Exp1 then op = Exp2 else op = Exp3;
- Exemplo:
  1. x = 10;
  2. y = x > 9 ? 100: 200;
  3. Ao final y contém o valor 100.
- Como seria o código com if-else?



Operador Vírgula

- O operador vírgula é usado para encadear expressões.
- O lado esquerdo de um operador vírgula é sempre avaliado como void [10].
- x = (y=3, y+1);
- Logo, atribui-se 3 a y, em seguida atribui-se 4 a x.
- Os parêntesis são necessários devido a precedência da atribuição que é maior.

[10] Assim, o lado direito torna-se a expressão do lado direito

Precedências

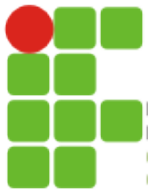
Precedência	Operador
Maior	() [ ] −>
	! ~ ++ −− − (*tipo*) * & *sizeof*
	* / %
	+ −
	<< >>
	== !=
	&
	∧
	!
	&&
	!!
	(? :)
	= += −= *= /= *etc*
Menor	,

Expressões

- Expressões são compostas de operadores, constantes e variáveis.
- C ANSI não estipula a ordem de avaliação das subexpressões.
- Compilador fica livre para produzir o melhor código.
- x = f1() + f2();
- f1() ou f2() são avaliadas primeiro? Depende do compilador.
- Constantes e variáveis de tipos diferentes é uma mesma expressão, elas são convertidas a um mesmo tipo.
- Converte todos os operandos no tipo do maior operando [11].
- Casts convertem uma expressão em um tipo.
- double x = (double) 10/3;

[11] Promoção de tipo.

- Notas de aula: adaptado de prof. Ajalmar Rocha.
- Livro Base: C Completo e Total. Herbert Schildt. Capítulo 2.



Obrigado!!!