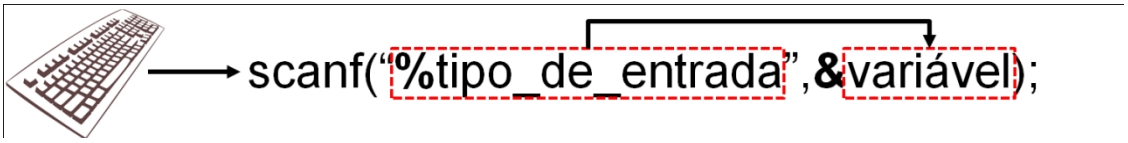
Tópico 3

Comando de entrada

scanf()

Comando que realiza a leitura dos dados da entrada padrão (no caso o teclado)

scanf(“tipo de entrada”, lista de variáveis)



O tipo de entrada deve ser sempre definido entre **“aspas duplas”**

```
int main(){
    int x;
    scanf("%d",&x);

    printf("Valor de x: %d",x);

    return 0;
}
```

Especificadores de formato do tipo de entrada

Alguns tipos de saída	
%c	leitura de um caractere (**char**)
%d ou %i	leitura de números inteiros (**int** ou **char**)
%f	leitura de número reais (**float** ou **double**)
%s	leitura de vários caracteres

Podemos ler mais de um valor em um único comando

Quando digitar vários valores, separar com espaço, TAB, ou Enter.

Exemplo:

```c
int main(){
    int x,z;
    float y;
    //Leitura de um valor inteiro
    scanf("%d",&x);
    //Leitura de um valor real
    scanf("%f",&y);
    //Leitura de um valor inteiro e outro real
    scanf("%d%f",&x,&y);
    //Leitura de dois valores inteiros
    scanf("%d%d",&x,&z);
    //Leitura de dois valores inteiros com espaço
    scanf("%d %d",&x,&z);

    return 0;
}
```

getchar()

Comando que realiza a leitura de um único caractere.

```c
int main(){
    char c;
    c = getchar();
    printf("Caractere: %c\n", c);
    printf("Codigo ASCII: %d\n", c);

    return 0;
}
```

Escopo de variáveis

São três escopos:

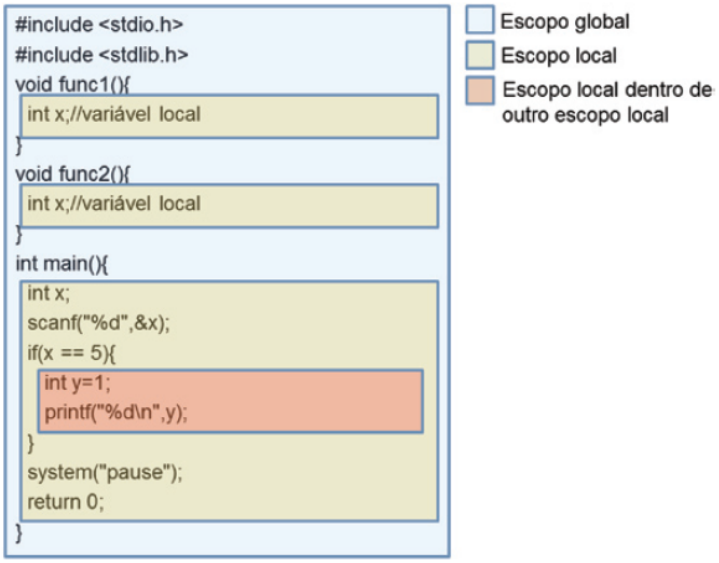
- Escopo
  Define onde e quando a variável pode ser usada.
- Escopo global
  Fora de qualquer definição de função
  Tempo de vida é o tempo de execução do programa
- Escopo local
  Bloco ou função

Escopo local

- Bloco: visível apenas no interior de um bloco de comandos
- Função: declarada na lista de parâmetros da função ou definida dentro da função

```
//bloco
if(x == 10){
     int i;
     i = i + 1;
}
//função
int soma (int x, int y) {
     int z;
     z = x + y;
     return z;
}
```

Observem as diferenças entre os três tipos de escopo:



Constantes

Como uma variável, uma constante também armazena um valor na memória do computador. Entretanto, esse valor não pode ser alterado: é constante. Para constantes é obrigatória a atribuição do valor.

Usando #define

Você deverá incluir a diretiva de pré-processador #define antes de início do código:
Cuidado: não colocar ";"
#define PI 3.1415

Usando const

Usando const, a declaração não precisa estar no início do código

A declaração é igual a de uma variável inicializada.

Ex:
const double pi = 3.1415;

Sequências de escape

São constantes predefinidas, elas permitem o envio de caracteres de controle não gráficos para dispositivos de saída.

Código	Comando
\a	som de alerta (bip)
\b	retrocesso (backspace)
\n	nova linha (new line)
\r	retorno de carro (carriage **return**)
\v	tabulação vertical
\t	tabulação horizontal
\'	apóstrofe
\"	aspa
\\	barra invertida (backslash)
\f	alimentação de folha (form feed)
\?	símbolo de interrogação
\0	caractere nulo (cancela a escrita do restante)

Exemplo:

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

int main(){
    printf("Hello World\n");
    printf("Hello\nWorld\n");
    printf("Hello \\ World\n");
    printf("\"Hello World\"\n");

    return 0;
}
```

Saída:

```
Hello World
Hello
World
Hello \ World
"Hello World"
```

Tipos Booleanos em C

Um tipo booleano pode assumir dois valores:

verdadeiro ou falso (true ou false)

Na linguagem C não existe o tipo de dado booleano. Para armazenar esse tipo de informação, use-se uma variável do tipo int (número inteiro)

Valor 0 significa falso

Números + ou – : verdadeiro

Exemplos:

```c
int AssentoOcupado = 1; // verdadeiro
int PortaAberta = 0; // falso
```

Operadores

Os operadores são usados para desenvolver diferentes tipos de operações. Com eles podemos:

- Realizar operações matemáticas com suas variáveis.
- Realizar operações de comparação entre suas variáveis.
- Realizar operações lógicas entre suas variáveis.
- Realizar operações em nível de bits com suas variáveis

Operadores aritméticos

São aqueles que operam sobre números (valores, variáveis, constantes ou chamadas de funções) e/ou expressões e têm como resultados valores numéricos.
Note que os operadores aritméticos são sempre usados em conjunto com o operador de atribuição.

Operador	Significado	Exemplo
+	Adição de dois valores	z = x + y
-	Subtração de dois valores	z = x – y
*	Multiplicação de dois valores	z = x * y
/	Quociente de dois valores	z = x / y
%	Resto de uma divisão	z = x % y

Podemos devolver o resultado para uma outra variável ou para um outro comando ou função que espere receber um valor do mesmo tipo do resultado da operação, no caso, a função **printf()**

```c
int main(){
    int x = 10, y = 20, z;

    z = x * y;
    printf("z = %d\n",z);

    z = y / 10;
    printf("z = %d\n",z);

    printf("x+y = %d\n",x+y);

    return 0;
}
```

IMPORTANTE

As operações de multiplicação, divisão e resto são executadas antes das operações de adição e subtração. Para forçar uma operação a ser executada antes das demais, ela é colocada entre parênteses

z = x * y + 10;
z = x * (y + 10);

O operador de subtração também pode ser utilizado para inverter o sinal de um número

x = −y;

Neste caso, a variável x receberá o valor de y multiplicado por −1, ou seja,

x = (−1) * y;

Em uma operação utilizando o operador de quociente /, se o numerador e o denominador forem números inteiros, por padrão o compilador retornará apenas a parte inteira da divisão

```c
int main(){
    float x;
    x = 5/4; // x = 1.000000
    printf("x = %f\n",x);

    x = 5/4.0; // x = 1.250000
    printf("x = %f\n",x);

    return 0;
}
```

Operadores relacionais

São aqueles que verificam a magnitude (qual é maior ou menor) e/ou igualdade entre dois valores e/ou expressões.

Os operadores relacionais são operadores de comparação de valores

Retorna verdadeiro (1) ou falso (0)

Operador	Significado	Exemplo
>	Maior do que	X > 5
>=	Maior ou igual a	X >= Y
<	Menor do que	X < 5
<=	Menor ou igual a	X <= Z
==	Igual a	X == 0
!=	Diferente de	X != Y

OBS: IMPORTANTE

*Símbolo de atribuição = é diferente, muito diferente, do operador relacional de igualdade ==

```c
int Nota;
Nota == 60; // Nota é igual a 60?
Nota = 50; // Nota recebe 50
// Erro comum em C:
// Teste se a nota é 60
// Sempre entra na condição
if (Nota = 60) {
    printf("Você passou raspando!!");
}
// Versão Correta
if (Nota == 60) {
    printf("Você passou raspando!!");
}
```

Por que sempre entra na condição?

```c
if (Nota = 60) {
    printf("Você passou raspando!!");
}
```

Ao fazer Nota = 60 (“Nota recebe 60”) estamos atribuindo um valor inteiro à variável Nota.

O valor atribuído 60 é diferente de Zero. Como em C os booleanos são números inteiros, então vendo Nota como booleano, essa assume true, uma vez que é diferente de zero

Operadores lógicos

Certas situações não podem ser modeladas utilizando apenas os operadores aritméticos e/ou relacionais

- Um exemplo bastante simples disso é saber se determinada variável x está dentro de uma faixa de valores.
- Por exemplo, a expressão matemática

  $0 < x < 10$

- indica que o valor de x deve ser maior do que 0 (zero) e também menor do que 10

- Os operadores lógicos permitem representar situações lógicas unindo duas ou mais expressões relacionais simples em uma composta

Retorna verdadeiro (1) ou falso (0)

Exemplo

A expressão $0 < x < 10$
Equivale a $(x > 0)$ && $(x < 10)$

Operador	Significado	Exemplo
&&	Operador **E**	(x > 0) && (x < 10)
||	Operador **OU**	(a == 'F') || (b != 32)
!	Operador **NEGAÇÃO**	!(x == 10)

Tabela Verdade

Os termos a e b representam o resultado de duas expressões relacionais

a	b	!a	!b	a && b	a || b
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

Outro Exemplo de Tabela Verdade:

			Operador		
			&& (.e.)	|| (.ou.)	! (.não.)
Expressão algoritmo	A = 5	B <> 9	A = 5 .e. B <> 9	A = 5 .ou. b <> 9	.não. A = 5
Expressão em C	A == 5	B != 9	A== 5 && B != 9	A == 5 || B != 9	!A == 5
Resultados possíveis	.V.	.V.	.V.	.V.	.F.
	.V.	.F.	.F.	.V.	.F.
	.F.	.V.	.F.	.V.	.V.
	.F.	.F.	.F.	.F.	.V.

A Tabela-verdade pode ser usada para facilitar a análise da combinação de expressões, variáveis e operadores lógicos. Na tabela acima podemos verificar que as expressões, A= 5 e B != 9, podem assumir quatro possibilidades, ou seja, ambas podem ser verdadeiras (primeira linha dos resultados possíveis), a

primeira pode ser verdadeira e a segunda falsa, a primeira pode ser falsa e a segunda verdadeira ou ambas podem ser falsas. Essas combinações dependem dos valores atribuídos as variáveis.

Exemplo:

```c
int main(){
    int r, x = 5, y = 3;
    r = (x > 2) && (y < x);//verdadeiro (1)
    printf("Resultado: %d\n",r);
    r = (x%2==0) && (y > 0);//falso (0)
    printf("Resultado: %d\n",r);
    r = (x > 2) || (y > x);//verdadeiro (1)
    printf("Resultado: %d\n",r);
    r = (x%2==0) || (y < 0);//falso (0)
    printf("Resultado: %d\n",r);
    r = !(x > 2);// falso (0)
    printf("Resultado: %d\n",r);
    r = !(x > 7) && (x > y);// verdadeiro (1)
    printf("Resultado: %d\n",r);

    return 0;
}
```

Operadores de pré e pós-incremento/decremento

Esses operadores podem ser utilizados sempre que for necessário somar uma unidade (incremento) ou subtrair uma unidade (decremento) a determinado valor.

Operador	Significado	Exemplo	Resultado
++	incremento	++x ou x++	x = x + 1
--	decremento	--x ou x--	x = x - 1

Essa diferença de sintaxe no uso do operador não tem importância se o operador for usado sozinho, porém se utilizado dentro de uma expressão aritmética, a diferença entre os dois operadores será evidente

```c
int main(){
    int x, y;
    x = 10;
    y = x++;
    printf("%d \n",x);// 11
    printf("%d \n",y);// 10

    y = ++x;
    printf("%d \n",x);// 12
    printf("%d \n",y);// 12

    return 0;
}
```

Operadores de atribuição simplificada

Muitos operadores são sempre usados em conjunto com o operador de atribuição. Para tornar essa tarefa mais simples, a linguagem C permite simplificar algumas expressões

Operador	Significado	Exemplo		
+=	Soma e atribui	x += y	igual a	x = x + y
-=	Subtrai e atribui	x -= y	igual a	x = x – y
*=	Multiplica e atribui	x *= y	igual a	x = x * y
/=	Divide e atribui o quociente	x /= y	igual a	x = x / y
%=	Divide e atribui o resto	x %= y	igual a	x = x % y

Exemplo sem Operador:

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int x = 10,y = 20;
    x = x + y - 10;
    printf("x = %d\n",x);
    x = x - 5;
    printf("x = %d\n",x);
    x = x * 10;
    printf("x = %d\n",x);
    x = x / 15;
    printf("x = %d\n",x);

    return 0;
}
```

Exemplo com operador

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int x = 10,y = 20;
    x += y - 10;
    printf("x = %d\n",x);
    x -= 5;
    printf("x = %d\n",x);
    x *= 10;
    printf("x = %d\n",x);
    x /= 15;
    printf("x = %d\n",x);

    return 0;
}
```

Exercícios

Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```c
int x, y, z;
x = y = 10;
z = ++x;
x -= x;
y++;
x = x + y - (z--);
```

Diga se as seguintes expressões serão verdadeiras ou falsas:

```c
int x = 7;
(x > 5) || (x > 10)
(!(x == 6) && (x >= 6)
```

Conversões de Tipos na Atribuição

O compilador converte automaticamente o valor do lado direto para o tipo do lado esquerdo do operador de atribuição “=” pode haver perda de informação.

Exemplo:

```
int x = 65;
char ch;
float f = 25.1;
//ch recebe 8 bits menos significativos de x
//converte para a tabela ASCII
ch = x;
printf("ch = %c\n",ch);// 'A'
//x recebe parte apenas a parte inteira de f
x = f;
printf("x = %d\n",x);// 25
//f recebe valor 8 bits convertido para real
f = ch;
printf("f = %f\n",f);// 65.000000
//f recebe o valor de x
f = x;
printf("f = %f\n",f);// 25.000000
```

Modeladores (Casts)

Quando um modelador é aplicado a uma expressão ele força o resultado da expressão a ser de um tipo especificado.

- (tipo) expressão

Exemplo:

```
float x, y, f = 65.5;

x = f / 10.0;
y = (int) (f / 10.0);
printf("x = %f\n",x);//6.550000
printf("y = %f\n",y);//6.000000
```

Precedência dos Operadores

MAIOR PRECEDÊNCIA	
++ --	Pré-incremento/decremento
()	Parênteses (chamada de função)
[ ]	Elemento de array
.	Elemento de struct
->	Conteúdo de elemento de ponteiro para struct
++ --	Pós-incremento/decremento
+ -	Adição e subtração unária
! ~	Não lógico e complemento bit a bit
(tipo)	Conversão de tipos (*type cast*)
*	Acesso ao conteúdo de ponteiro
&	Endereço de memória do elemento
sizeof	Tamanho do elemento
* / %	Multiplicação, divisão e módulo (resto)
+ -	Adição e subtração
<< >>	Deslocamento de bits à esquerda e à direita
< <=	"Menor do que" e "menor ou igual a"
> >=	"Maior do que" e "maior ou igual a"
== !=	"Igual a" e "diferente de"
&	E bit a bit
^	OU exclusivo
|	OU bit a bit
&	E lógico
||	OU lógico
?:	Operador ternário
=	Atribuição
+= -=	Atribuição por adição ou subtração
*= /= %=	Atribuição por multiplicação, divisão ou módulo (resto)
<<= >>=	Atribuição por deslocamento de bits
&= ^= | =	Atribuição por operações lógicas
,	Operador vírgula
MENOR PRECEDÊNCIA	