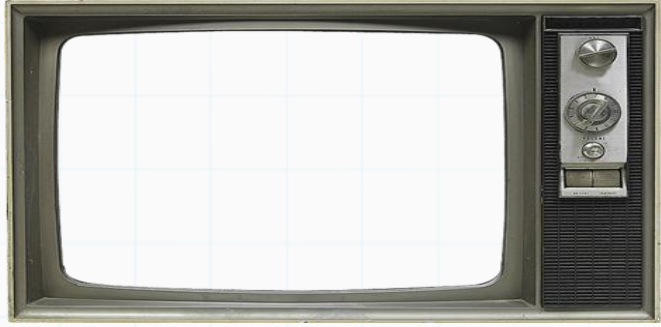
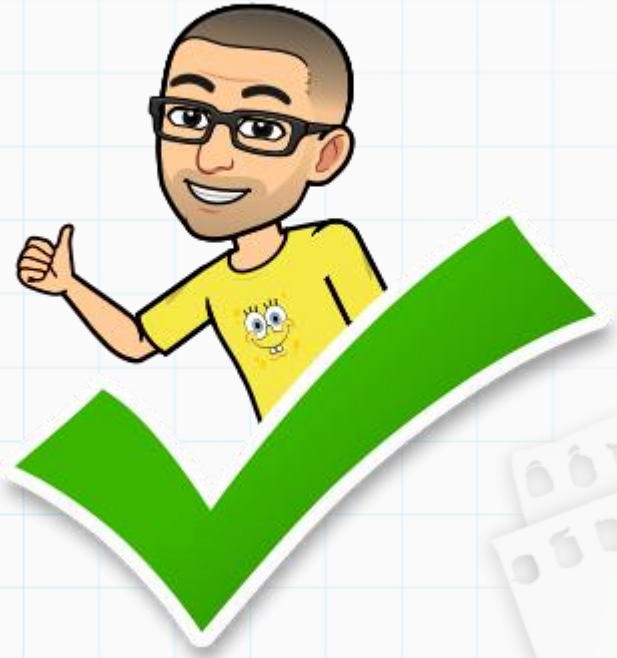
Programação Estruturada

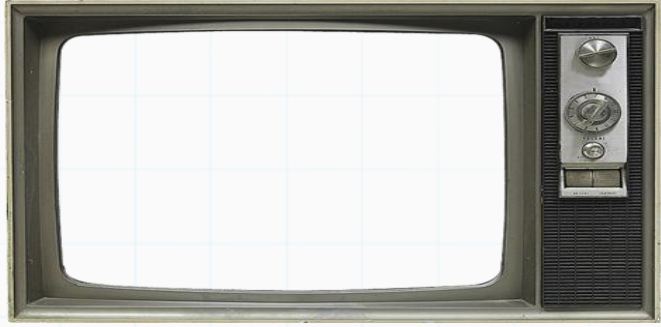


Professor : Yuri Frota

yuri@ic.uff.br



Introdução



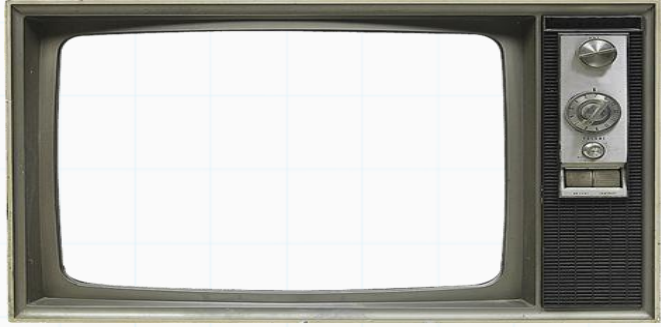
Estrutura de um programa em C:

- um programa em C sempre começa a sua execução a partir da função main

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

int main()
{
    /* sequencia de comandos */
    return 0;
}
```

- a blocagem (definição de início e fim da estrutura) em C é feito com “{ }“

Introdução



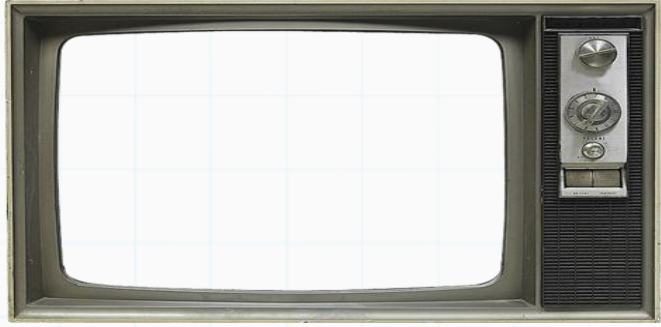
Estrutura de um programa em C:

- um programa em C sempre começa a sua execução a partir da função main

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

int main()
{
    /* sequencia de comandos */
    return 0;
}
```

- a blocagem (definição de início e fim da estrutura) em C é feito com “{ }“
- Indentações não importam como no Python, mas ajuda na compreensão.

Introdução



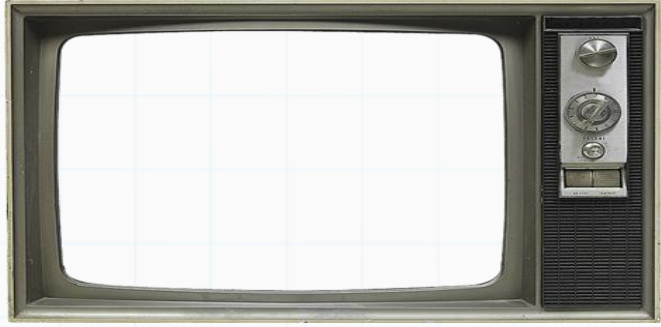
Estrutura de um programa em C:

- um programa em C sempre começa a sua execução a partir da função main

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

int main()
{
    /* sequencia de comandos */
    return 0;
}
```

- a blocagem (definição de início e fim da estrutura) em C é feito com “{ }“
- Indentações não importam como no Python, mas ajuda na compreensão.
- Comentários são feitos com /* ... */ para várias linhas e // para uma linha

Introdução



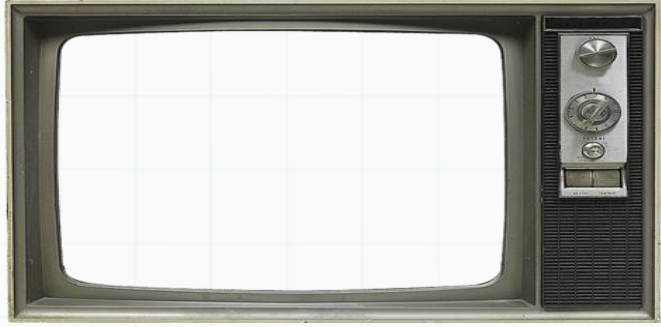
Estrutura de um programa em C:

- um programa em C sempre começa a sua execução a partir da função main

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

int main()
{
    /* sequencia de comandos */
    return 0;
}
```

- a blocagem (definição de início e fim da estrutura) em C é feito com “{ }“
- Indentações não importam como no Python, mas ajuda na compreensão.
- Comentários são feitos com /* ... */ para várias linhas e // para uma linha
- Bibliotecas padrão são inseridas no início do programa com o comando #include <...>

Introdução



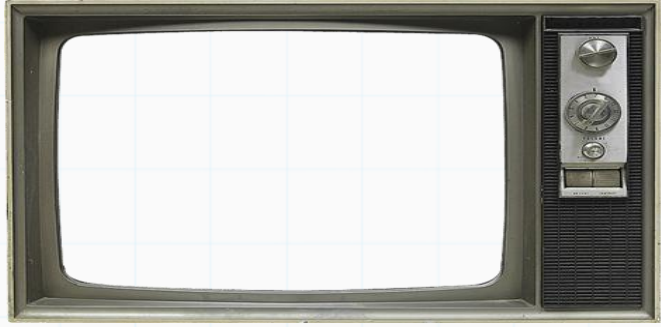
Estrutura de um programa em C:

- um programa em C sempre começa a sua execução a partir da função main

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

int main()
{
    /* sequencia de comandos */
    return 0;
}
```

- a blocagem (definição de início e fim da estrutura) em C é feito com “{ }“
- Indentações não importam como no Python, mas ajuda na compreensão.
- Comentários são feitos com /* ... */ para várias linhas e // para uma linha
- Bibliotecas padrão são inseridas no início do programa com o comando #include <...>
- Maiúsculas e Minúsculas são diferentes

Introdução



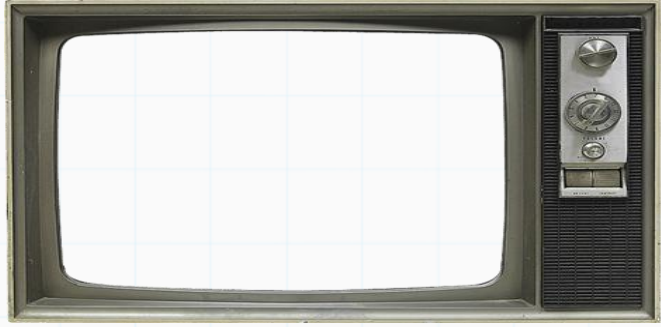
Estrutura de um programa em C:

- um programa em C sempre começa a sua execução a partir da função main

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

int main()
{
    /* sequencia de comandos */
    return 0;
}
```

- a blocagem (definição de início e fim da estrutura) em C é feito com “{ }“
- Indentações não importam como no Python, mas ajuda na compreensão.
- Comentários são feitos com /* ... */ para várias linhas e // para uma linha
- Bibliotecas padrão são inseridas no início do programa com o comando #include <...>
- Maiúsculas e Minúsculas são diferentes
- Todos os comandos (não estruturas) tem que ser delimitadas no fim por “;”

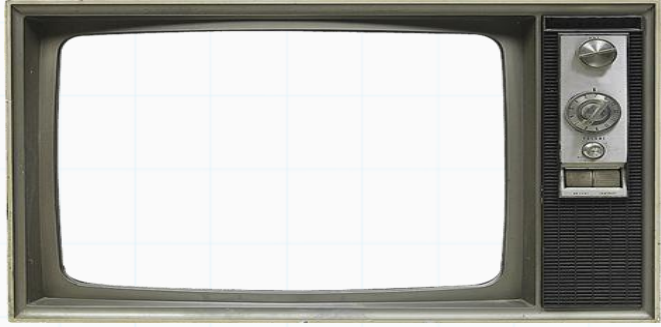
Introdução



- Variáveis:
  - Toda variável tem:
    - um nome (identificador)
    - um tipo de dado
    - um valor
  - Restrição para nomes: não é permitido começar o nome com um algarismo (0-9), alguns caracteres não são válidos (*, -, /, +, ...), e palavras reservadas não podem ser utilizadas (main, if, while, ...)

```
pikachu = 10;
nome = 'p';
```

Introdução

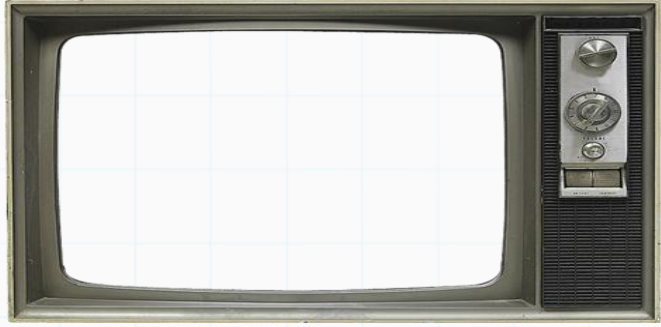


- Variáveis:
  - Toda variável tem:
    - um nome (identificador)
    - um tipo de dado
    - um valor
  - Restrição para nomes: não é permitido começar o nome com um algarismo (0-9), alguns caracteres não são válidos (*, -, /, +, ...), e palavras reservadas não podem ser utilizadas (main, if, while, ...)
  - É necessário informar o nome e o tipo das nossas variáveis:
    - O compilador precisa saber o tipo do dado para reservar o espaço de memória definido para aquele tipo (quantidade de bytes).

```
int pikachu;
char nome;
float p1,p2;
double a3;
```

```
pikachu = 10;
nome = 'p';
```

Introdução

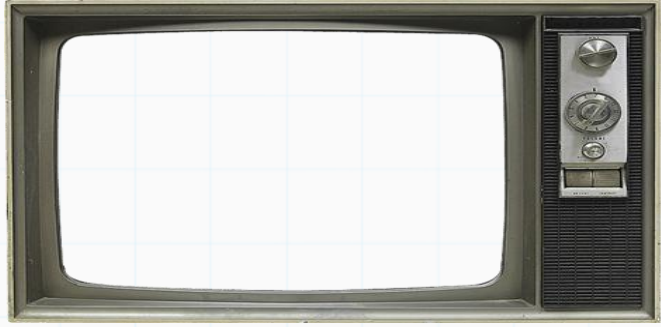


- Variáveis:
  - Toda variável tem:
    - um nome (identificador)
    - um tipo de dado
    - um valor
  - Restrição para nomes: não é permitido começar o nome com um algarismo (0-9), alguns caracteres não são válidos (*, -, /, +, ...), e palavras reservadas não podem ser utilizadas (main, if, while, ...)
  - É necessário informar o nome e o tipo das nossas variáveis:
    - O compilador precisa saber o tipo do dado para reservar o espaço de memória definido para aquele tipo (quantidade de bytes).

- Constantes: Não podem ser alteradas -> “#define”

```
#define PI 3.1415
#define MIL 10000
```

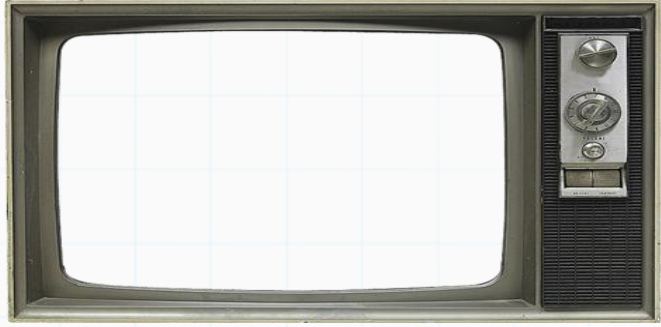
Introdução



- Tipo de Dados:
  - Existem poucos tipos de dados básicos em C
    - char: guarda um caractere
    - int: guarda um inteiro
    - float, double: guardam números reais

```
int pikachu;
char nome;
float p1,p2;
double a3;
```

Introdução



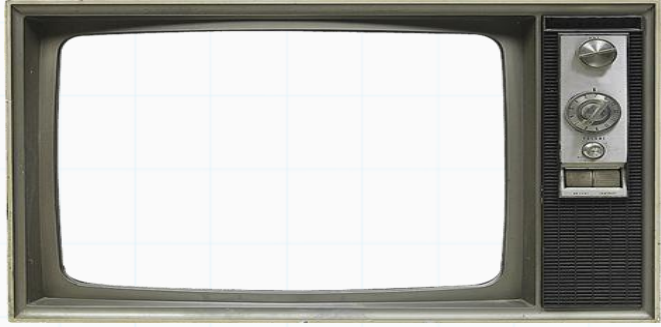
- Tipo de Dados:
  - Existem poucos tipos de dados básicos em C
    - char: guarda um caractere
    - int: guarda um inteiro
    - float, double: guardam números reais

  - Um tipo básico pode ter seu significado alterado por um modificador:
    - short, long, signed, unsigned

```
int pikachu;
char nome;
float p1,p2;
double a3;
```

```
short int pequeno_int;
long int grande_int;
```

Introdução



- Tipo de Dados:
  - Existem poucos tipos de dados básicos em C
    - char: guarda um caractere
    - int: guarda um inteiro
    - float, double: guardam números reais

```
int pikachu;
char nome;
float p1,p2;
double a3;
```

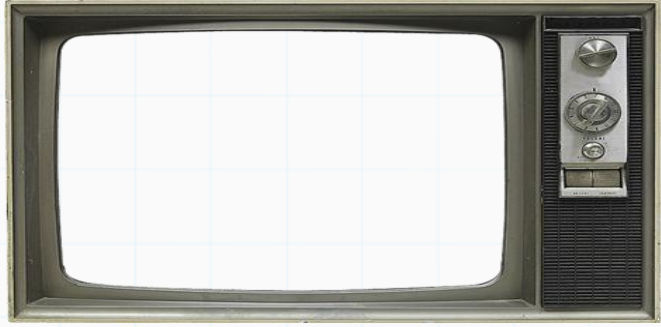
  - Um tipo básico pode ter seu significado alterado por um modificador:
    - short, long, signed, unsigned

```
short int pequeno_int;
long int grande_int;
```

  - Não existe tipo lógico em C (0->False e 1->True) Alguns compiladores aceitam "_Bool"

```
int boleano;
```

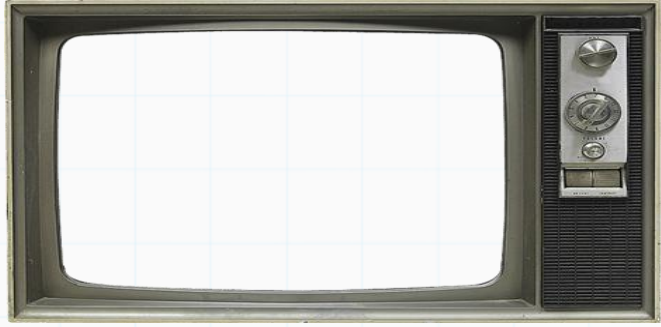
Introdução



- Tipo de Dados:
  - Existem poucos tipos de dados básicos em C
    - char: guarda um caractere
    - int: guarda um inteiro
    - float, double: guardam números reais
  - Um tipo básico pode ter seu significado alterado por um modificador:
    - short, long, signed, unsigned
  - Não existe tipo lógico em C (0->False e 1->True) Alguns compiladores aceitam "_Bool"

Tipo	Tamanho	Representatividade
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
short int	2 bytes	-32768 a 32767
unsigned short int	2 bytes	0 a 65535
long int	4 bytes	-2.147.483.648 a 2.147.483.647
unsigned long int	4 bytes	0 a 4.294.967.295
int	Depende da máquina	
float	4 bytes	-10^{-38} a 10^{38}
double	8 bytes	-10^{-308} a 10^{308}

Introdução

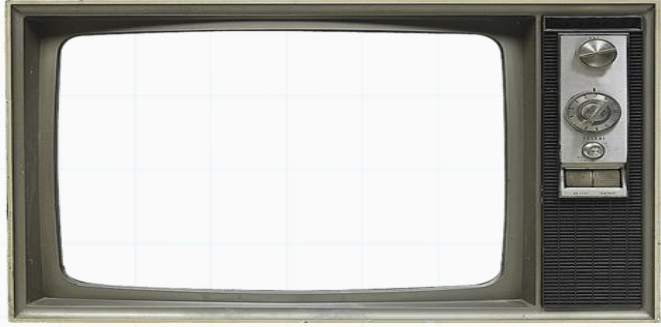


- Declaração de Variáveis:
  - Toda variável tem que ser declarada e inicializada antes de ser usada

```
float p1,p2;

p1 = 10;
p2 = 0.3;
```

Separados

Introdução



- Declaração de Variáveis:
  - Toda variável tem que ser declarada e inicializada antes de ser usada

```
float p1,p2;

p1 = 10;
p2 = 0.3;
```

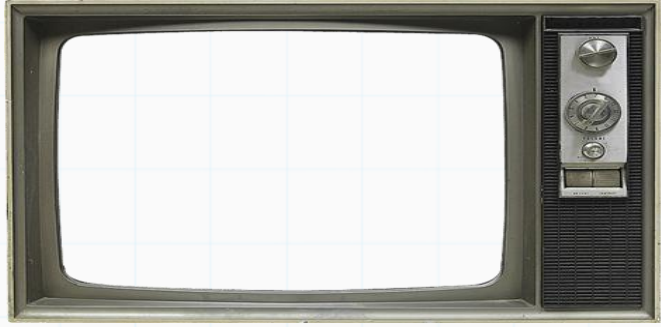
Separados

```
float p1=10;
char letra='a';
char palavra[10] = "goku";
```

Juntos

Aspas simples definem o caractere, e duplas definem strings

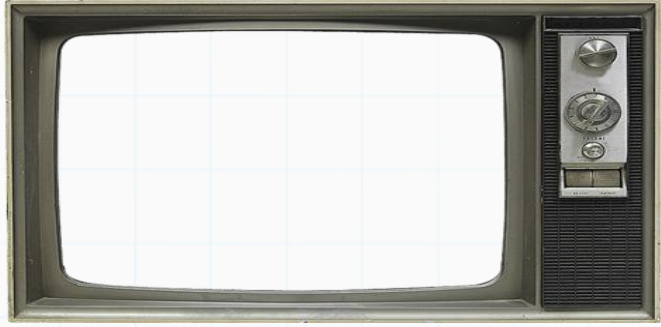
Introdução



- Escopo de Variáveis:
  - Variáveis podem ser LOCAIS ou GLOBAIS
  - Uma variável local só é reconhecida dentro do bloco “{ }” onde ela foi declarada
    - Vive enquanto a função está sendo executada
    - Nenhuma outra função tem acesso a ela
    - Para uma melhor prática de programação, as variáveis locais devem ser declaradas no início de uma função

```
int main()
{
    int var_local;


    return 0;
}
```

Introdução

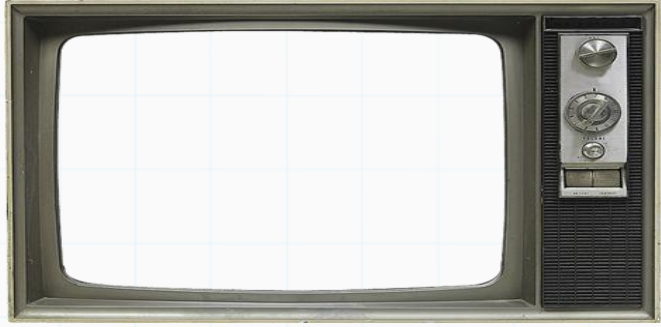


- Escopo de Variáveis:
  - Variáveis podem ser LOCAIS ou GLOBAIS
  - Uma variável local só é reconhecida dentro do bloco “{ }” onde ela foi declarada
    - Vive enquanto a função está sendo executada
    - Nenhuma outra função tem acesso a ela
    - Para uma melhor prática de programação, as variáveis locais devem ser declaradas no início de uma função

  - Uma variável global é reconhecida no programa inteiro
    - Definidas no início do código fonte
    - Acessadas por quaisquer funções
    - Existem permanentemente e retêm seus valores armazenados

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

int G1, G2;
int teste ( ) {
        int C, D;
        ...
}
int main () {
          int count;
          ...
}
```

Introdução



- Operadores Aritméticos:
  - \+ Adição
  - \- Subtração
  - * Multiplicação
  - / Divisão
  - % Resto (Só pode ser usado com inteiro)
  - = Atribuição

```
int a=5;
int b=2;
int c;
float d;

c=a/b;
d=a/b;
```

a/b é 2
c é 2
d é 2.0

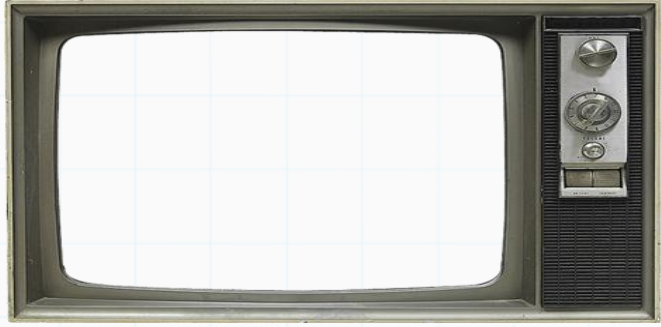
int/int = int
float/int = float
int/float = float
float/float = float

```
int a=5;
float b=2;
int c;
float d;

c=a/b;
d=a/b;
```

a/b é 2.5
c é 2
d é 2.5

Precedência entre operadores é praticamente igual ao do Python (esquerda para direita)

Introdução

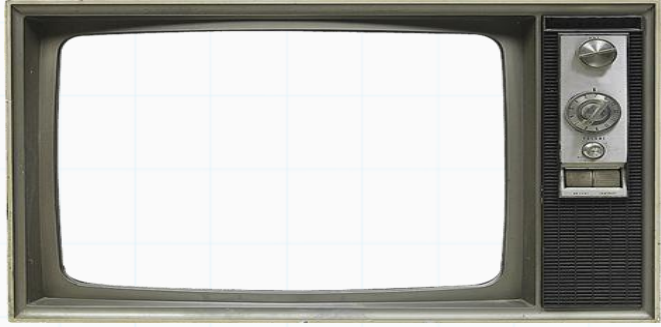


Operadores de Atribuição:

Considere que inicialmente
`cont = 0;`

+=	soma e atribuição	`cont += 4;`	`cont = cont + 4;`	cont→4
-=	subtração e atribuição	`cont -= 2;`	`cont = cont - 2;`	cont→2
*=	multiplicação e atribuição	`cont *= 3;`	`cont = cont * 3;`	cont→6
/=	divisão e atribuição	`cont /= 2;`	`cont = cont / 2;`	cont→3

Introdução



Operadores de Incremento/Decremento:

Incremento e Decremento

`++` → operador de incremento, soma 1 ao operando

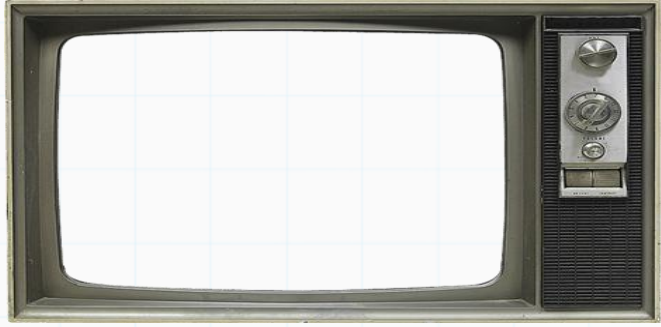
`--` → operador de decremento, subtrai 1 do operando

Os dois operadores podem ser pré-fixados (antes da variável) ou pós-fixados (depois da variável)

Exemplo:

```
int n;
n++;
++n;
```

Introdução



Operadores de Incremento/Decremento:

Incremento e Decremento

`++` → operador de incremento, soma 1 ao operando

`--` → operador de decremento, subtrai 1 do operando

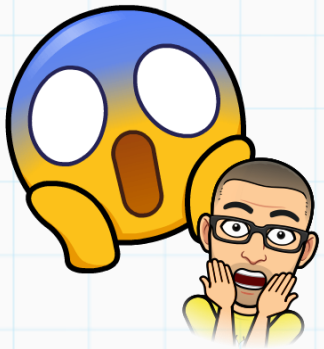
Os dois operadores podem ser pré-fixados (antes da variável) ou pós-fixados (depois da variável)

MAS CUIDADO !!!

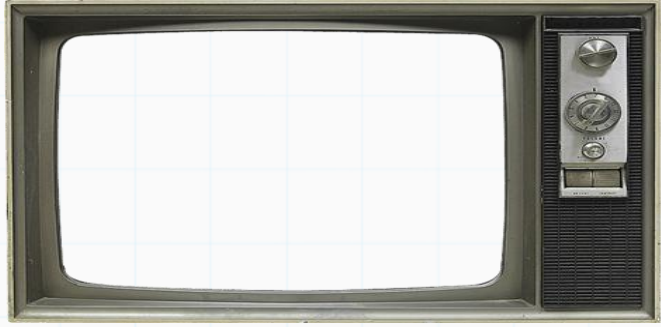
Suponha as expressões, onde `n = 5`

```
x = n++;
```

→ `x` recebe o valor de `n`, e depois `n` é incrementado
→ então `x = 5` e `n = 6`



Introdução



Operadores de Incremento/Decremento:

Incremento e Decremento

`++` → operador de incremento, soma 1 ao operando

`--` → operador de decremento, subtrai 1 do operando

Os dois operadores podem ser pré-fixados (antes da variável) ou pós-fixados (depois da variável)

MAS CUIDADO !!!

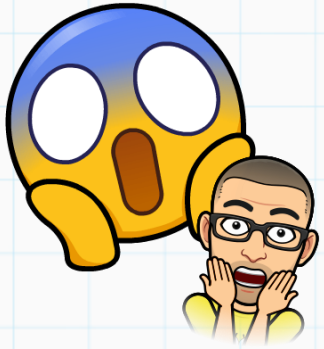
Suponha as expressões, onde `n = 5`

```
x = n++;
```

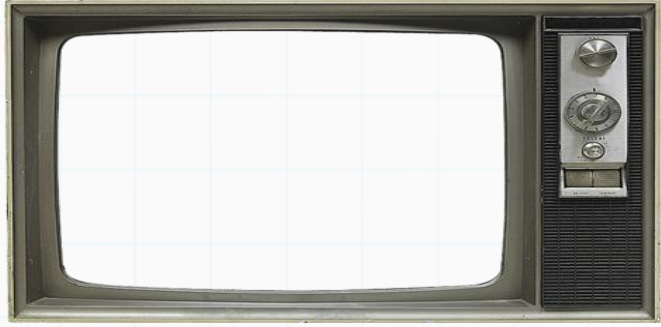
→ `x` recebe o valor de `n`, e depois `n` é incrementado
→ então `x = 5` e `n = 6`

```
x = ++n;
```

→ `x` recebe o valor de `n`, já incrementado
→ então `x = 6` e `n = 6`



Introdução

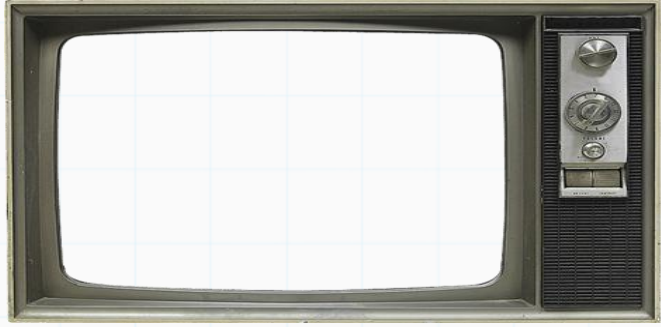


Operadores de Incremento/Decremento:

- Podemos então escrever de diversas formas o incremento

```
a = a + 1;
a++;
++a;
a += 1;
```

Introdução



Operadores de Incremento/Decremento:

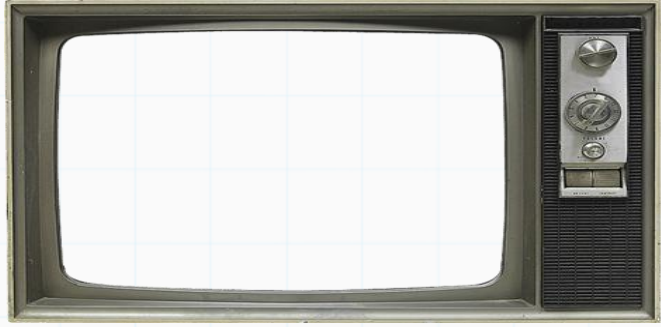
- Podemos então escrever de diversas formas o incremento

```
a = a + 1;
a++;
++a;
a += 1;
```

- E atribuições

```
i += 2;        → i = i + 2;

x /= y;        → x = x / y;

x *= y + 1;    → x = x * (y + 1);   CERTO!
               → x = x * y + 1;     ERRADO!
```

Introdução



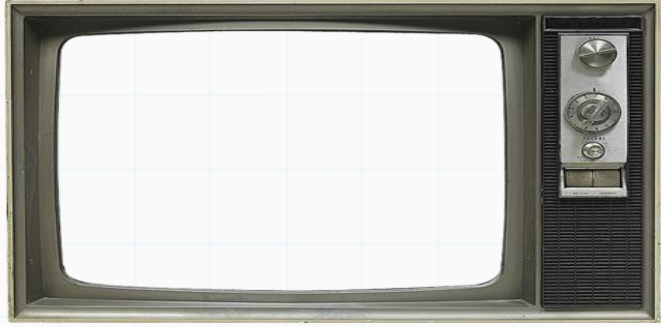
Operadores Relacionais:

Símbolo	Nome do Operador	Exemplo	Significado
>	Maior que	x > y	x é maior que y?
>=	Maior ou igual	x >= y	x é maior ou igual a y ?
<	Menor que	x < y	x é menor que y?
<=	Menor ou igual	x <= y	x é menor ou igual a y ?
==	Igualdade	x == y	x é igual a y?
!=	Diferente de	x != y	x é diferente de y?

Operadores relacionais possuem menor precedência que os aritméticos:

`i < lim - 1;` → equivale a `i < (lim - 1);`

Introdução



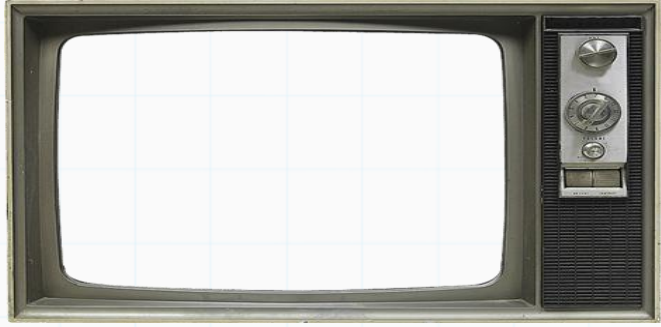
Operadores Lógicos:

& & → E

| | → OU

! → Não

Introdução



Operadores Lógicos:

&& → E

|| → OU

! → Não

A	B	A && B	A || B	!A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

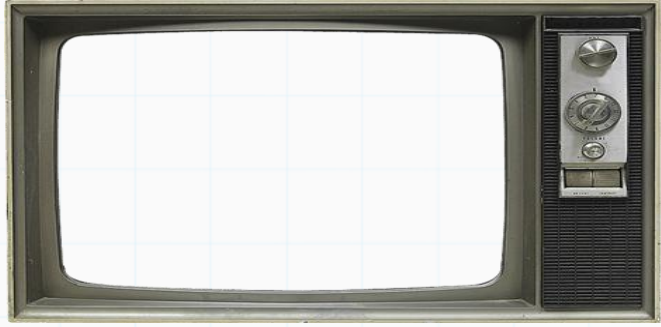
Na verdade !A vai ser 0 dado qualquer valor de A diferente de 0

Exemplos:

```
a = 3;
b = 5;
(a > b) && (b == 5)
!(a == 3 && b == 5) || (a < b)
```

qual é o valor ?

Introdução



Operadores Lógicos:

&& → E

|| → OU

! → Não

A	B	A && B	A || B	!A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

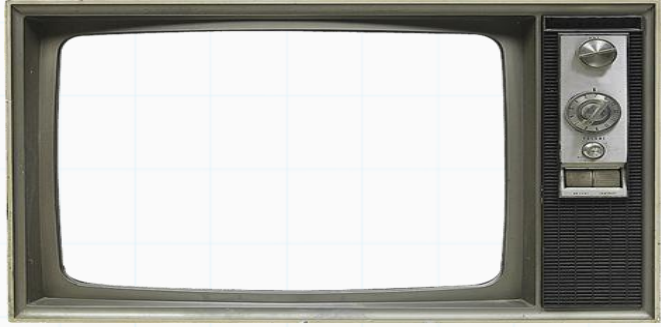
Na verdade !A vai ser 0 dado qualquer valor de A diferente de 0

Exemplos:

```
a = 3;
b = 5;
(a > b) && (b == 5)  -> 0
!(a == 3 && b == 5) || (a < b)  -> 1
```

qual é o valor ?

Introdução



Precedência entre operadores:

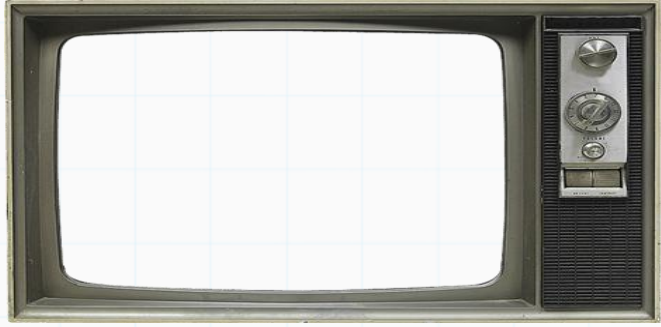
maior

menor

! ++ --	
% / *	Aritméticos e lógico
+ -	
< <= > >=	relacionais
== !=	
&&	lógicos
||	
= += -= /= %=	atribuição

Ficou na dúvida, usa parêntese

Introdução



Conversão entre tipos:

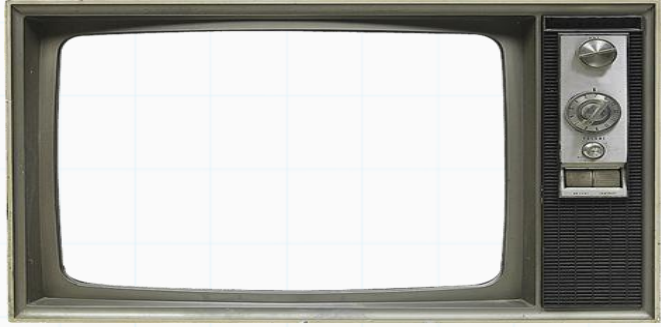
Exemplos de algumas conversões automáticas:

```
int a;
float y;

a = 9;
y = a;
```

→ y = 9.0

→ tipo “menor” atribuído ao “maior”

Introdução



Conversão entre tipos:

Exemplos de algumas conversões automáticas:

```
int a;
float y;

a = 9;
y = a;
```

→ y = 9.0

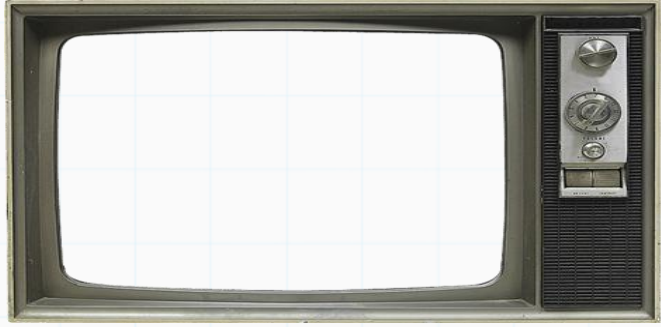
→ tipo “menor” atribuído ao “maior”

```
y = 3.5;
a = y;
```

→ a = 3

→ tipo “maior” é atribuído ao “menor” e ocorre perda da informação

Introdução

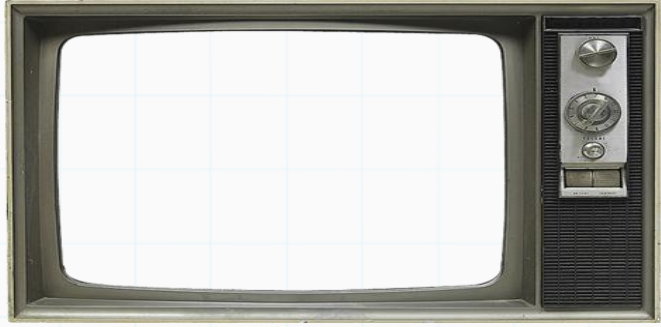


Conversão entre tipos:

Quando a expressão tiver operandos de tipos diferentes, o menor é convertido para o maior antes da operação.
A operação é feita na precisão do tipo mais representativo

```
int a;
double b, c;
a = 3.5;        → a = 3
b = a/2.0;      → b = 1.5
c = 1/3 +b;     → c = 0 + 1.5 → c = 1.5
c = 1/3.0 +b;   → c = 0.333 + 1.5 → c = 1.833
```

Introdução



Conversão entre tipos:

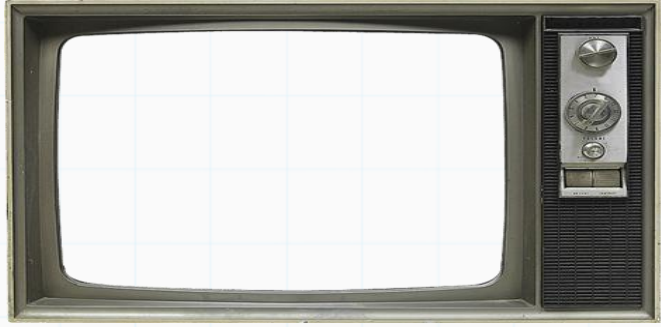
Quando a expressão tiver operandos de tipos diferentes, o menor é convertido para o maior antes da operação.
A operação é feita na precisão do tipo mais representativo

```
int a;
double b, c;
a = 3.5;        → a = 3
b = a/2.0;      → b = 1.5
c = 1/3 +b;     → c = 0 + 1.5 → c = 1.5
c = 1/3.0 +b;   → c = 0.333 + 1.5 → c = 1.833
```

O programador pode explicitamente requisitar uma conversão de tipo

```
int a;
double b;
a = (int) 3.5;          → a = 3
b = (double) 5          → b = 5.0
```

Introdução



Exemplos:

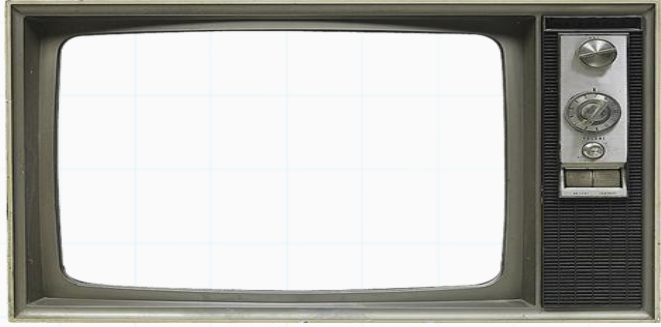
Considere:

```
int x, y, z;
int teste;
x = 5;
y = x++;
z = x--;
```

Quais os valores de `teste, x, y` e `z` depois da avaliação das seguintes expressões:

```
(a) teste = !y == !x;
(b) teste = ((x++ > y) || (--z <= y));
(c) teste = ((!x) || (!(!z)));
```

Introdução



Exemplos:

Considere:

```
int x, y, z;
int teste;
x = 5;
y = x++;
z = x--;
```

Quais os valores de `teste, x, y` e `z` depois da avaliação das seguintes expressões:

```
(a) teste = !y == !x;
(b) teste = ((x++ > y) || (--z <= y));
(c) teste = ((!x) || (!(!z)));
```

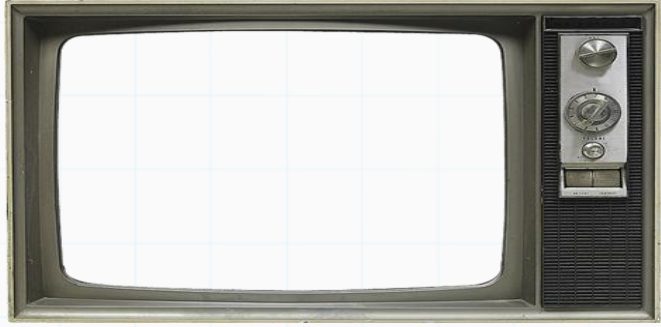
Solução

1

1

1

Introdução



Saida básica:

Comando printf imprime na tela

Formato

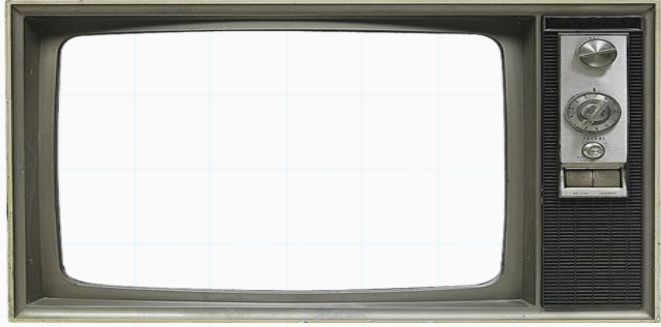
printf(“string formatada”, var1, var2, var....)

Bem parecido com python

Para usar `printf` deve ser usada a biblioteca `stdio.h`

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

Introdução



Saida básica:

Comando printf imprime na tela

Formato

printf(“string formatada”, var1, var2, var....)

Exemplos:

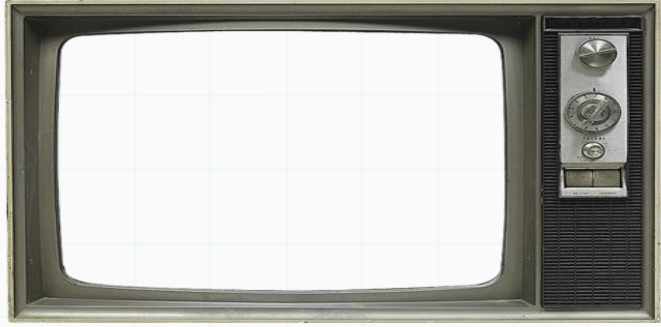
```
printf(“Primeiro Programa”);
printf(“\n”);
printf(“Primeiro Programa\n”);
printf(“O número inteiro é: %d”, num);
printf(“A soma de: %f + %f = %f”, a,b,c);
printf(“O resultado é: %.2f”, num);
```

Bem parecido com python

Para usar `printf` deve ser usada a biblioteca `stdio.h`

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

Introdução

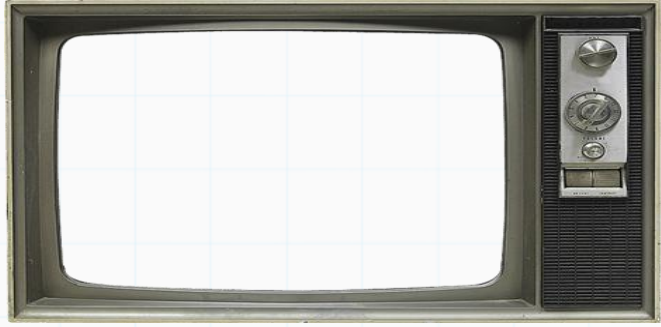


Saida básica:

Tipos de formatação

Formato	Especificação
%c	char
%d ou %i	int
%s	string
%f	float, double
%e	Notação cientifica
\n	nova linha
\”	“
\\	\
%%	%

Introdução



Entrada básica:

Comando scanf recebe informações do teclado

Formato

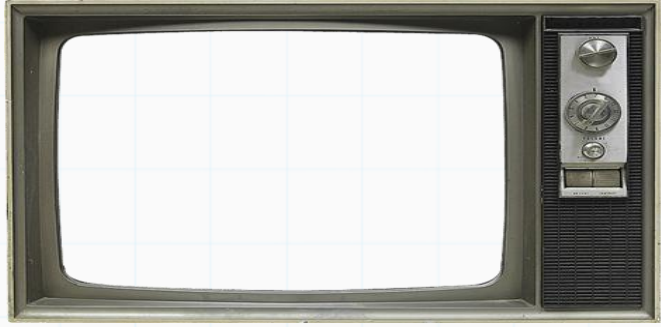
scanf(“string formatada de entrada”, &var1, &var2, &var....)

- Variáveis int, float e char são precedidas por “&” (ponteiro)
- Variáveis string não são precedidas por “&”

Para usar `scanf` deve ser usada a biblioteca `stdio.h`

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

Introdução



Entrada básica:

Comando scanf recebe informações do teclado

Formato

scanf("string formatada de entrada", &var1, &var2, &var....)

- Variáveis int, float e char são precedidas por "&" (ponteiro)
- Variáveis string não são precedidas por "&"

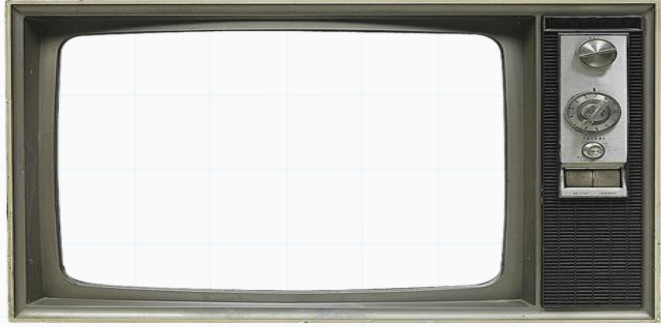
Para usar `scanf` deve ser usada a biblioteca `stdio.h`

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

```
float nota;

printf("Digite a nota da prova: ");
scanf("%f", &nota);
printf("nota = %f", nota);
```

```
Digite a nota da prova: 55.4
nota = 55.400002
```

Introdução



Entrada básica:

Comando scanf recebe informações do teclado

Formato

printf(“string formatada de entrada”, &var1, &var2, &var....)

- Variáveis int, float e char são precedidas por “&” (ponteiro)
- Variáveis string não são precedidas por “&”

Para usar `scanf` deve ser usada a biblioteca `stdio.h`

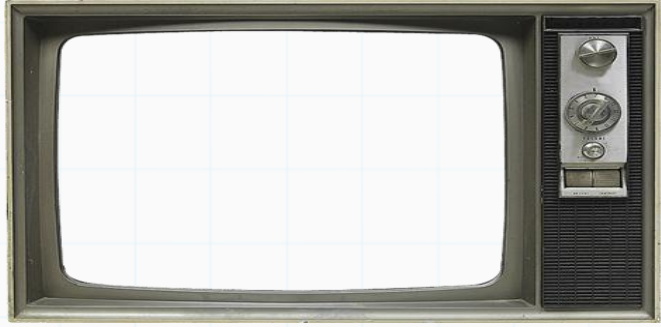
```
#include<stdio.h>
```

```
char str [80];
int i;

printf ("nome: ");
scanf ("%s",str);
printf ("idade: ");
scanf ("%d",&i);
printf ("Mr. %s , %d anos.\n",str,i);
```

```
nome: yuri
idade: 48
Mr. yuri , 48 anos.
```

Lendo strings

Introdução



Entrada básica:

Comando scanf recebe informações do teclado

Formato

printf(“string formatada de entrada”, &var1, &var2, &var....)

- Variáveis int, float e char são precedidas por “&” (ponteiro)
- Variáveis string não são precedidas por “&”

Para usar `scanf` deve ser usada a biblioteca `stdio.h`

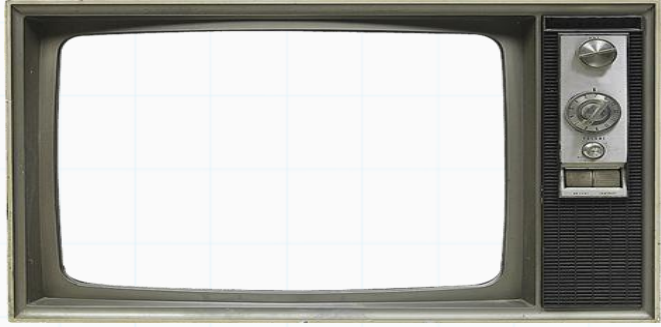
```
#include<stdio.h>
```

```
int a,b,c;

printf ("3 numeros inteiros: ");
scanf ("%d %d %d",&a,&b,&c);
printf ("%d %d %d",a,b,c);
```

```
3 numeros inteiros: 4 6 7
4 6 7
```

Lendo vários valores com um scanf

Introdução



Entrada básica:

Comando scanf recebe informações do teclado

Formato

printf(“string formatada de entrada”, &var1, &var2, &var....)

Para usar `scanf` deve ser usada a biblioteca `stdio.h`

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

- Variáveis int, float e char são precedidas por “&” (ponteiro)
- Variáveis string não são precedidas por “&”

```
int dia, mes, ano;
printf("qual data de hoje:");
scanf("%d/%d/%d", &dia, &mes, &ano);
printf("%d/%d/%d", dia, mes, ano);
```

Errado

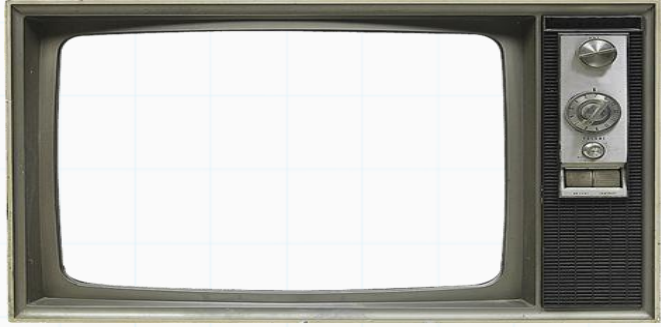
```
qual data de hoje:12 12 12
12/16/0
```

Certo

```
qual data de hoje:12/12/12
12/12/12
```

Cuidado com a formatação

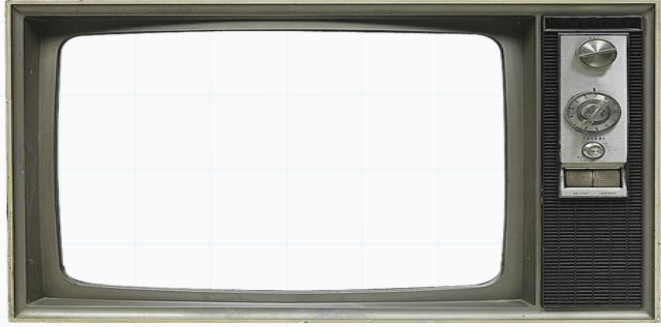
Introdução



Exemplo: programa que calcula o perímetro da circunferência com 2 casas decimais.

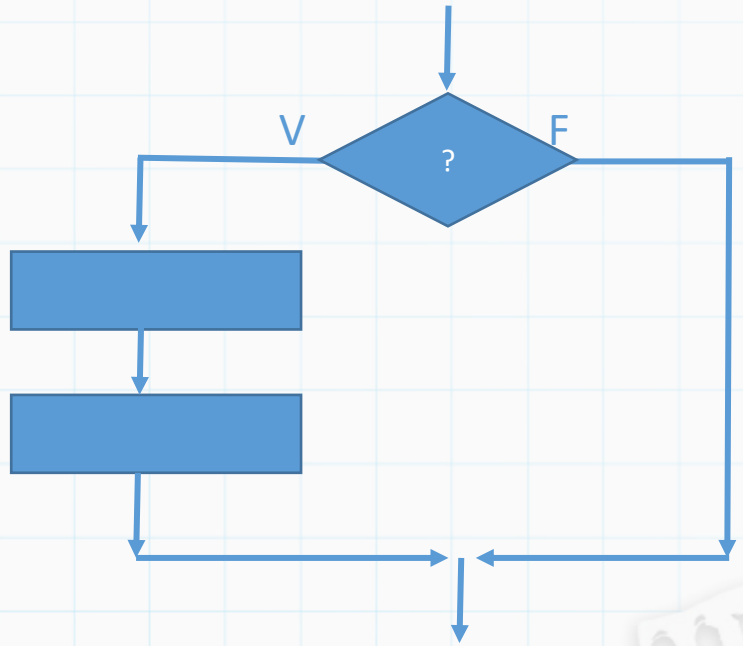
```c
#include <stdio.h>

#define pi 3.14

int main ()
{
  float raio;

  printf("Qual raio?");
  scanf("%f",&raio);

  printf("perimetro = %.2f", (2*pi*raio) );
  return 0;

}
```

Decisão

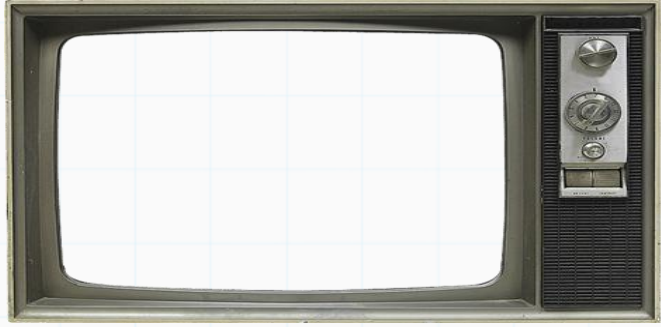


Comando IF:
Executa o bloco de instruções somente se a condição for verdadeira

```
Portugol

...
se CONDIÇÃO então
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
...
```



Decisão



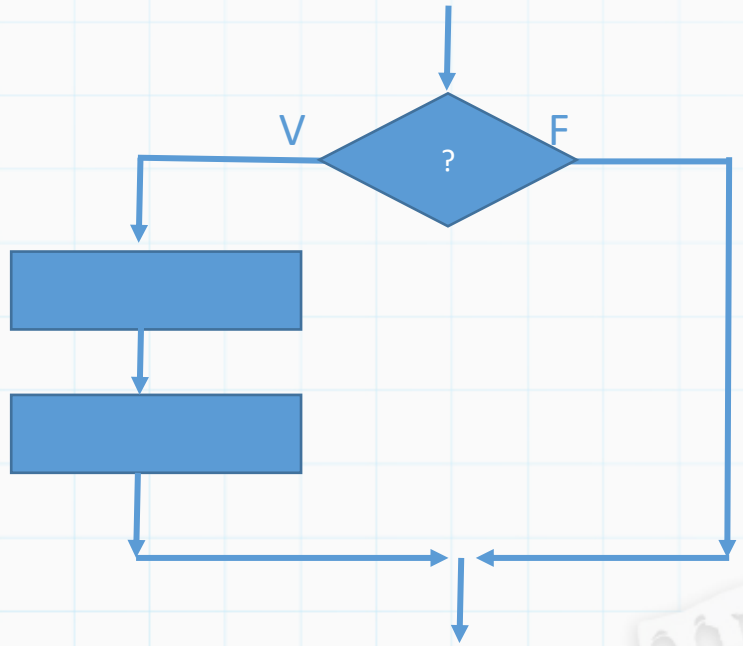
Comando IF:
Executa o bloco de instruções somente se a condição for verdadeira

Portugol

```
...
se CONDIÇÃO então
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
...
```

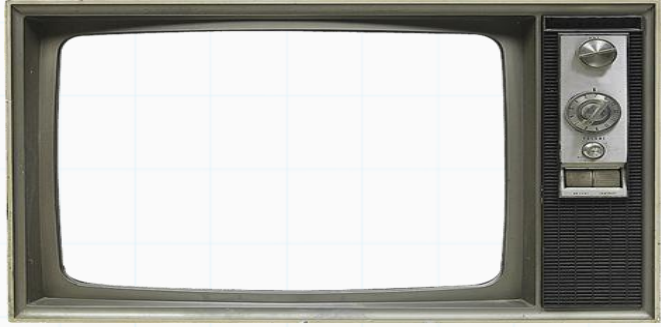
C

```
...
if (CONDIÇÃO)
   INSTRUÇÃO 1;
...

if (CONDIÇÃO)
{
   INSTRUÇÃO 1;
   INSTRUÇÃO 2;
   ...
   INSTRUÇÃO N;
}
```



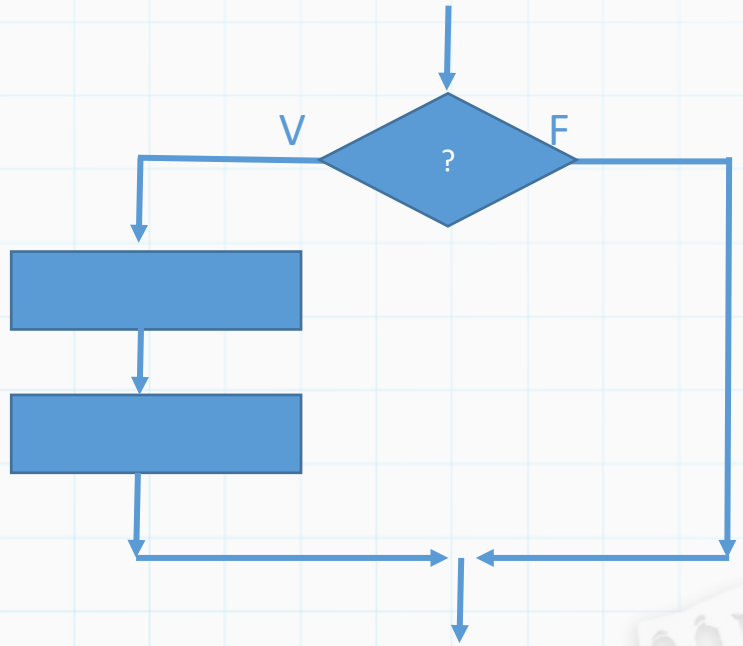
- Se tiver só uma instrução, não precisa de “{}”
- - o parêntese “()” é necessário para a condição

Decisão

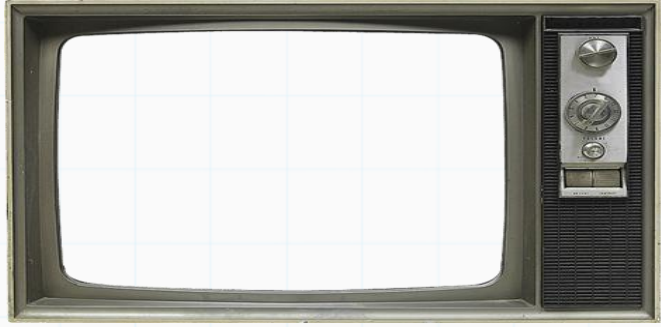


Comando IF:
Executa o bloco de instruções somente se a condição for verdadeira

```
if (20 > 18)
 printf("20 is greater than 18");

int x = 20;
int y = 18;
if (x > y)
{
  printf("x is greater than y");
  x=x+y;
}
```



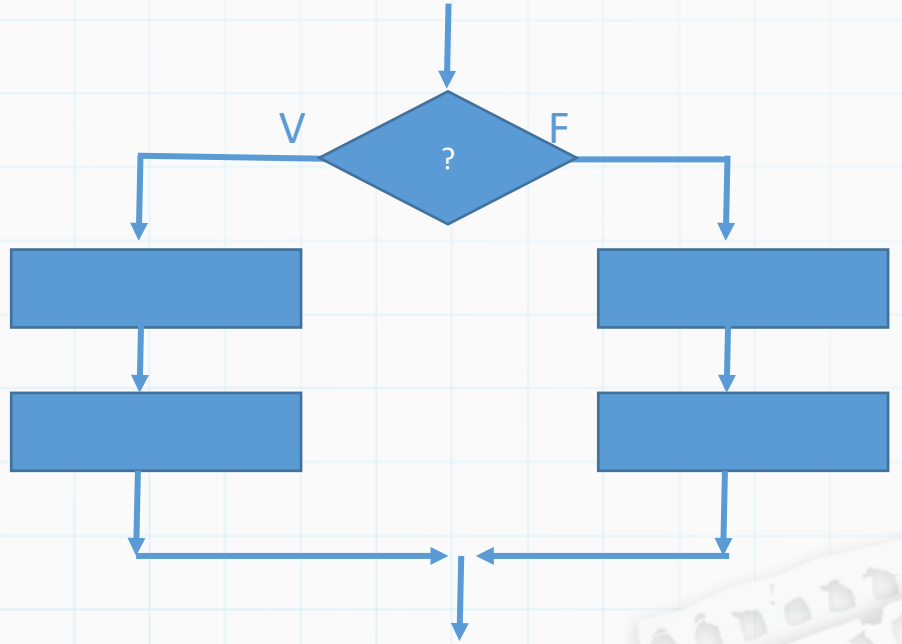
Decisão



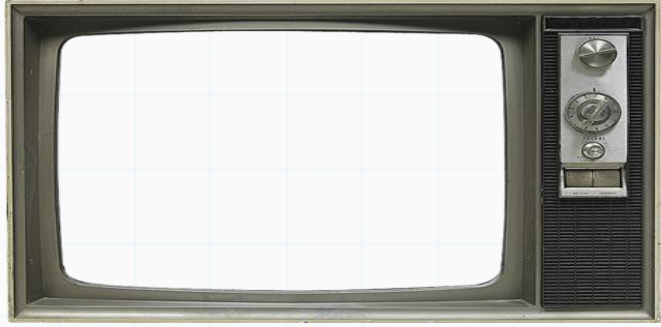
Comando IF ELSE:

Executa o bloco de instruções somente se a condição for verdadeira e outro caso seja falso

```
Portugol

...
Se CONDIÇÃO então
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
Senão
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
...
```



Decisão



Comando IF ELSE:

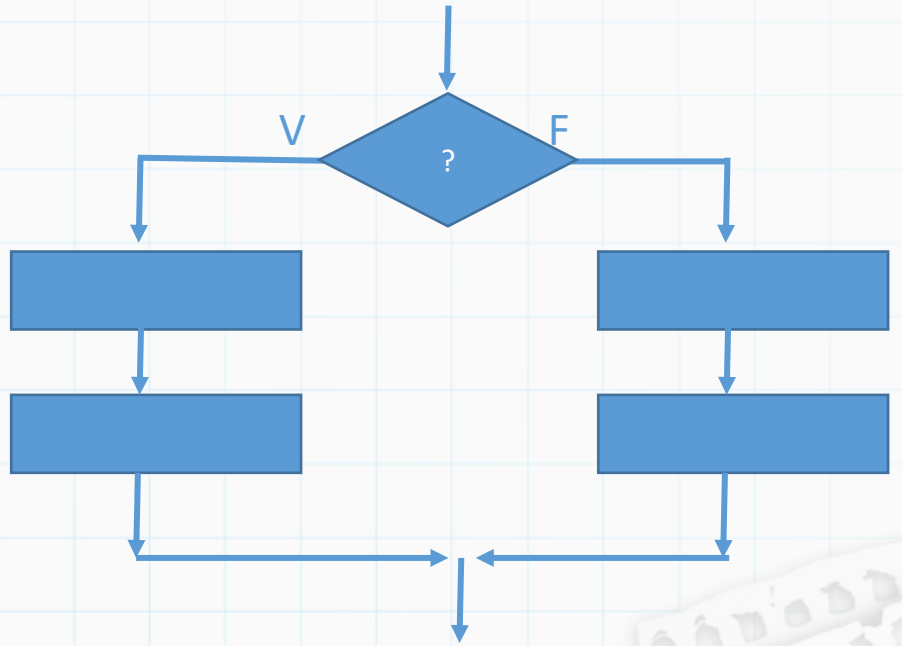
Executa o bloco de instruções somente se a condição for verdadeira e outro caso seja falso

Portugol

```
...
Se CONDIÇÃO então
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
Senão
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
...
```

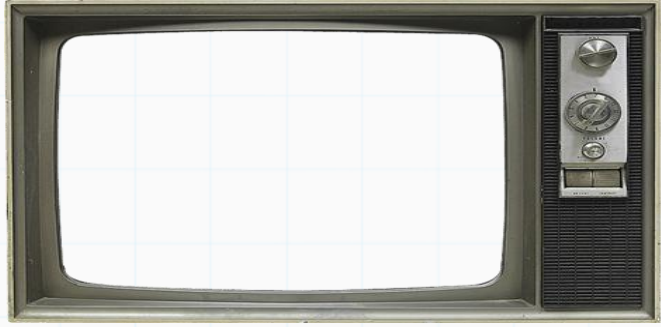
C

```
...
if (CONDIÇÃO)
{
   INSTRUÇÃO 1;
   INSTRUÇÃO 2;
   ...
   INSTRUÇÃO N;
}
else
{
   INSTRUÇÃO 1;
   INSTRUÇÃO 2;
   ...
   INSTRUÇÃO N;
}
...
```



- Se tiver só uma instrução, não precisa de “{}”
- - o parêntese “()” é necessário para a condição

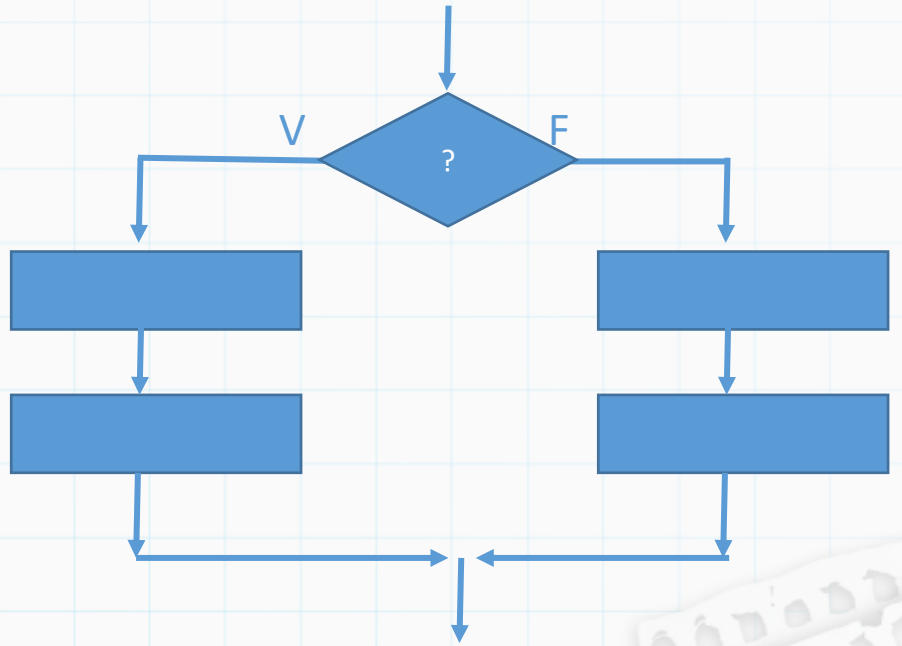
Decisão



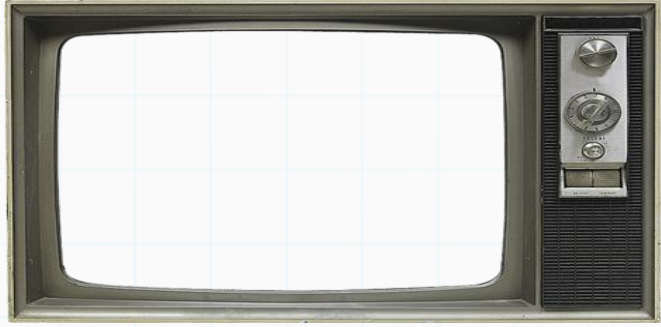
Comando IF ELSE:

Executa o bloco de instruções somente se a condição for verdadeira e outro caso seja falso

```c
int main() {
    int number;
    printf("numero: ");
    scanf("%d", &number);

    // True if the remainder is 0
    if (number%2 == 0) {
        printf("%d par.",number);
    }
    else {
        printf("%d impar.",number);
    }

    return 0;
}
```



Decisão



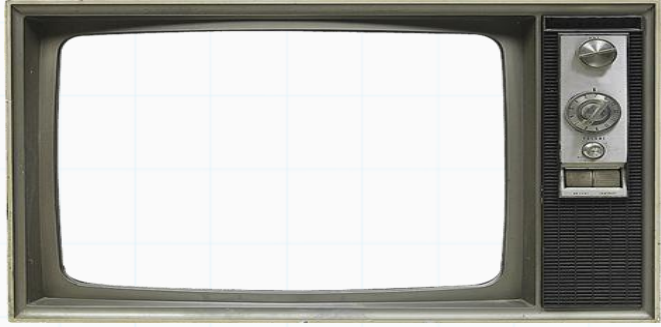
Encadeamento de IFs:

Sendo o ELSE opcional, pode existir “dúvida” quando ele é omitido em um encadeamento de IFs

```
int main() {
    if (n > 0)
     if (a > b)
          z = a;
      else
          z = b;
```

O ELSE pertence a que IF ?

Decisão



Encadeamento de IFs:

Sendo o ELSE opcional, pode existir “dúvida” quando ele é omitido em um encadeamento de IFs

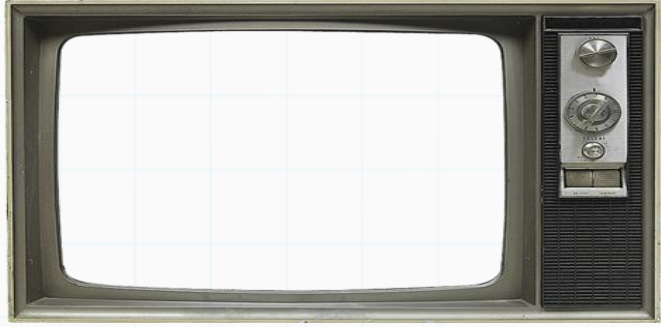
```
int main() {
    if (n > 0)
    if (a > b)
          z = a;
     else
          z = b;
```

O ELSE pertence a que IF ?
R: mais interno

```
int main() {
     if (n > 0)
     {
       if (a > b)
         z = a;
     }
     else
         z = b;
```

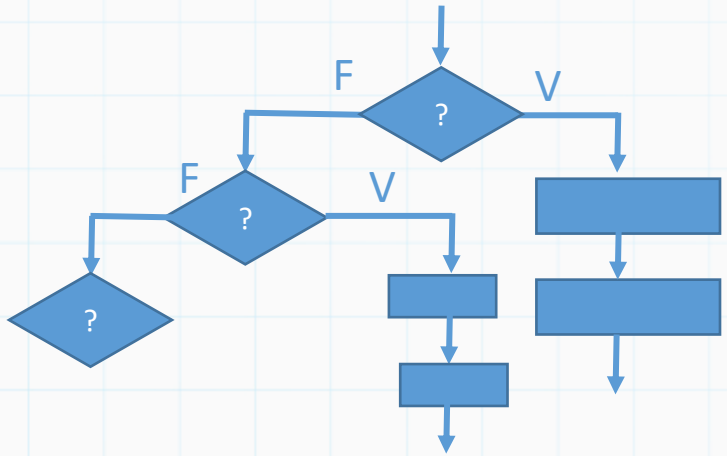
Para evitar ambiguidades, use “{}”

Decisão



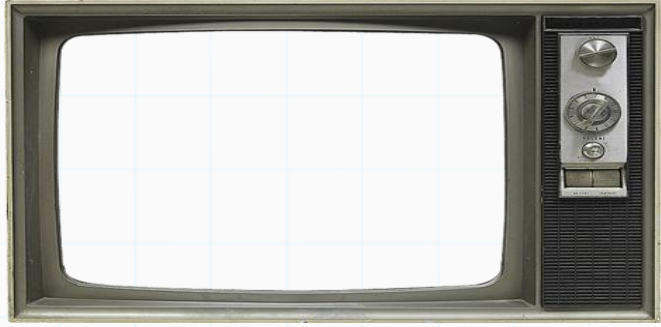
Comando IF ELSE IF:

Apenas o bloco no qual a condição é verdadeira é executado




```
Portugol

...
Se CONDIÇÃO então
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
Senão Se CONDIÇÃO então
                INSTRUÇÃO 1
                INSTRUÇÃO 2
                ...
                INSTRUÇÃO N
                ...
```

Decisão



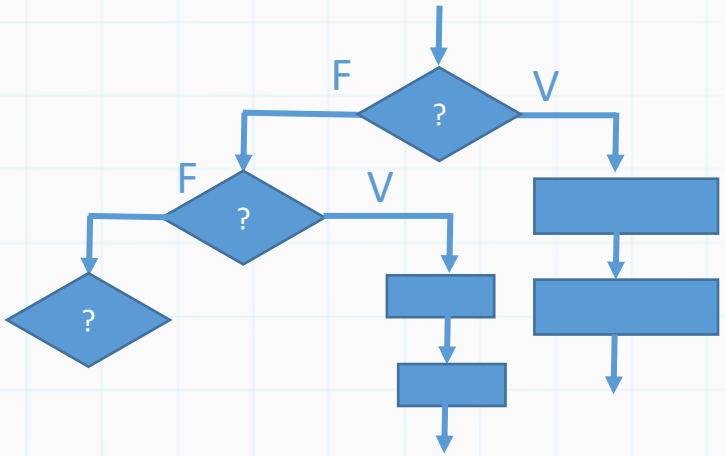
Comando IF ELSE IF:
Apenas o bloco no qual a condição é verdadeira é executado

Portugol

```
...
Se CONDIÇÃO então
   INSTRUÇÃO 1
   INSTRUÇÃO 2
   ...
   INSTRUÇÃO N
Senão Se CONDIÇÃO então
                INSTRUÇÃO 1
                INSTRUÇÃO 2
                ...
                INSTRUÇÃO N
                ...
```

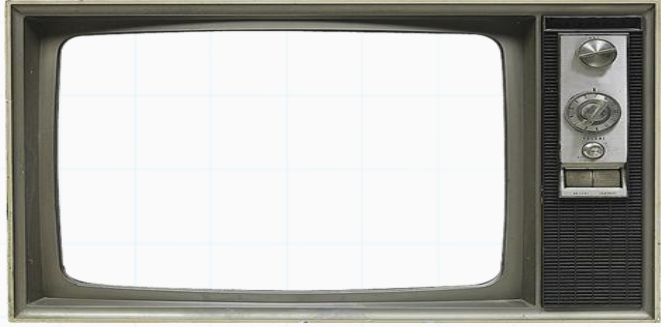
C

```
...
if (CONDIÇÃO)
{
   INSTRUÇÃO 1;
   INSTRUÇÃO 2;
   ...
   INSTRUÇÃO N;
}
else if (CONDIÇÃO)
{
   INSTRUÇÃO 1;
   INSTRUÇÃO 2;
   ...
   INSTRUÇÃO N;
}
   ...
```



- Se tiver só uma instrução, não precisa de “{}”
- - o parêntese “()” é necessário para a condição

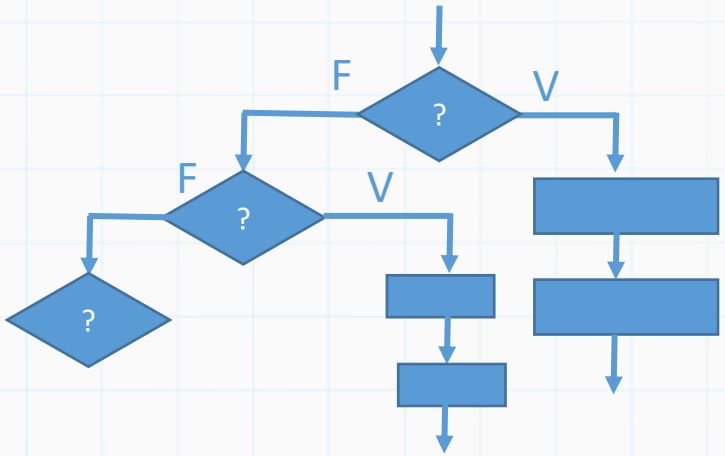
Decisão



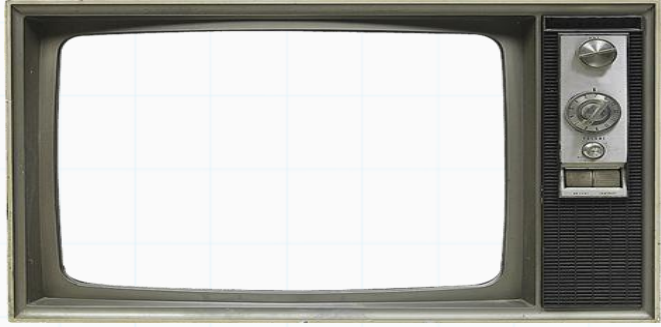
Comando IF ELSE IF:
Apenas o bloco no qual a condição é verdadeira é executado

```
  ...
if (CONDIÇÃO) {
   INSTRUÇÃO 1;
   ...
   INSTRUÇÃO N;
}
else if (CONDIÇÃO){
   INSTRUÇÃO 1;
   ...
   INSTRUÇÃO N;
}
else if (CONDIÇÃO) {
   INSTRUÇÃO 1;
   ...
   INSTRUÇÃO N;
}
else {
   ...
}
```

- É possível colocar tantos ***else if*** quantos forem necessários
- É possível adicionar um ***else*** ao final de tudo
  Nesse caso, se nenhuma condição for verdadeira, o bloco do **else** será executado



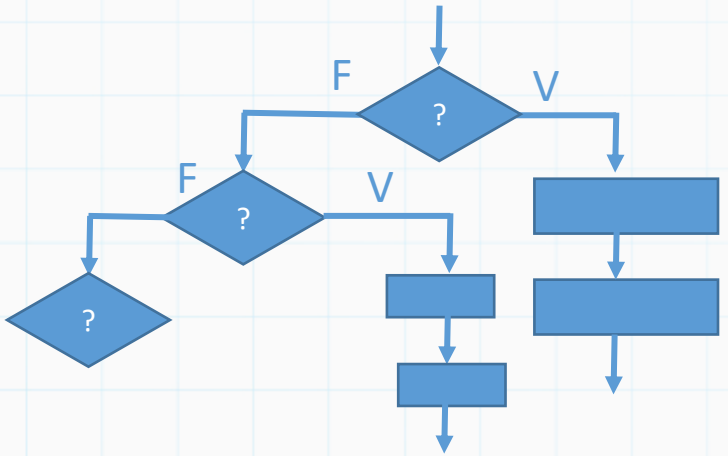
Decisão



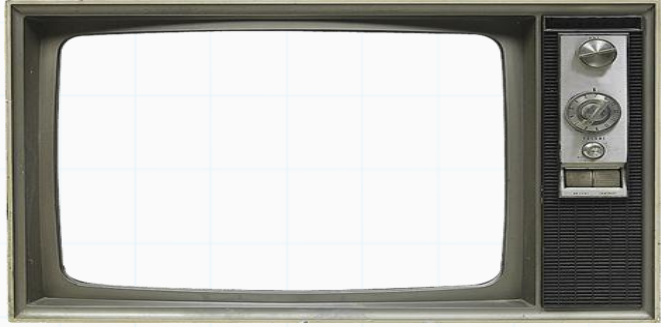
Comando IF ELSE IF:

Apenas o bloco no qual a condição é verdadeira é executado

```c
#include <stdio.h>
int main()
{
 int num;


     printf("Digite um número: \n");
     scanf("%d", &num);
     if(num==0){
        printf("O número é NULO\n");
     }
     else if(num<8){
        printf("O número é Menor que 8\n");
     }
     else if(num<70){
        printf("O número é maior que 70\n");
     }
     else{
        printf("O número está fora das condições criadas\n");
     }
  return(0);
/*   */
}
```



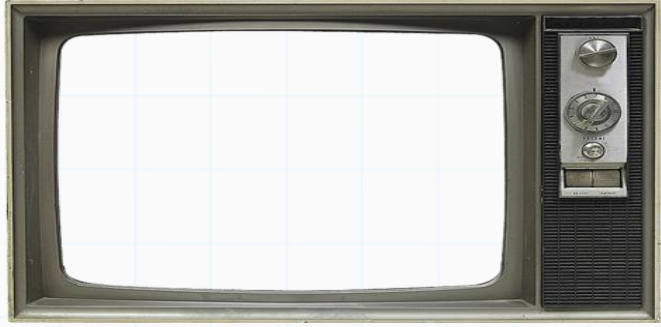
Decisão



Exemplos: O que será escrito ?

```c
#include <stdio.h>

    void main()

    {
        int x = 5;

        if (x < 1)
            printf("hello");
        if (x == 5)
            printf("hi");
        else
            printf("no");

    }
```

Decisão



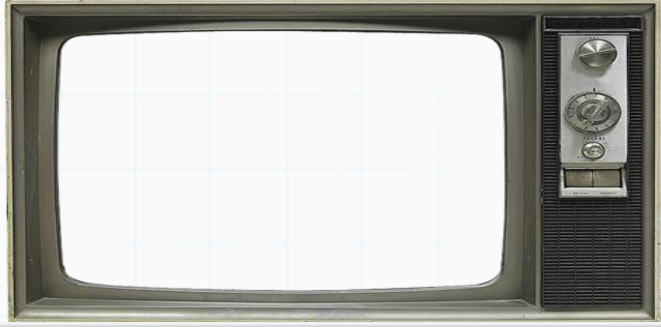
Exemplos: O que será escrito ?

```c
#include <stdio.h>

    void main()

    {
        int x = 5;

        if (x < 1)
            printf("hello");
        if (x == 5)
            printf("hi");
        else
            printf("no");

    }
```

hi

```c
#include <stdio.h>

    void main()
    {
        int x = 0;

        if (x == 0)
            printf("hi");
        else
            printf("how are u");
            printf("hello");
    }
```

Decisão



Exemplos: O que será escrito ?

```c
#include <stdio.h>

    void main()

    {
        int x = 5;

        if (x < 1)
            printf("hello");
        if (x == 5)
            printf("hi");
        else
            printf("no");

    }
```

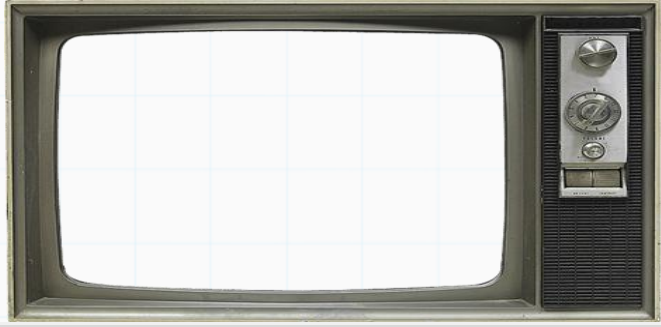
hi

```c
#include <stdio.h>

    void main()
    {
        int x = 0;

        if (x == 0)
            printf("hi");
        else
            printf("how are u");
            printf("hello");
    }
```

hihello

```c
#include <stdio.h>
int main()
{
    int x = 0;

    if (x++)
        printf("true\n");
    else if (x == 1)
        printf("false\n");
}
```

Decisão



Exemplos: O que será escrito ?

```c
#include <stdio.h>

    void main()

    {
        int x = 5;

        if (x < 1)
            printf("hello");
        if (x == 5)
            printf("hi");
        else
            printf("no");

    }
```

hi

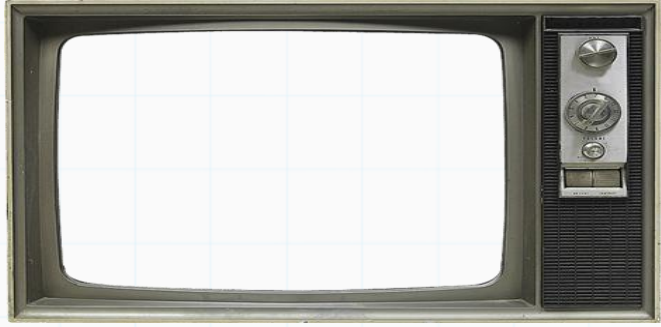
```c
#include <stdio.h>

    void main()
    {
        int x = 0;

        if (x == 0)
            printf("hi");
        else
            printf("how are u");
            printf("hello");
    }
```

hihello

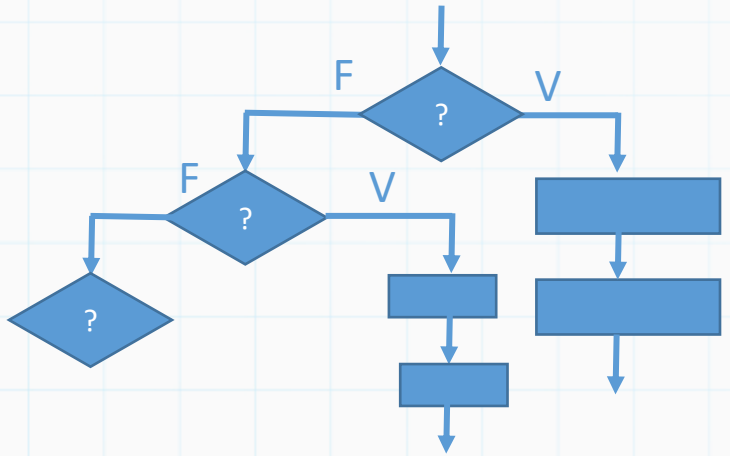
```c
#include <stdio.h>
int main()
{
    int x = 0;

    if (x++)
        printf("true\n");
    else if (x == 1)
        printf("false\n");
}
```

false

Decisão

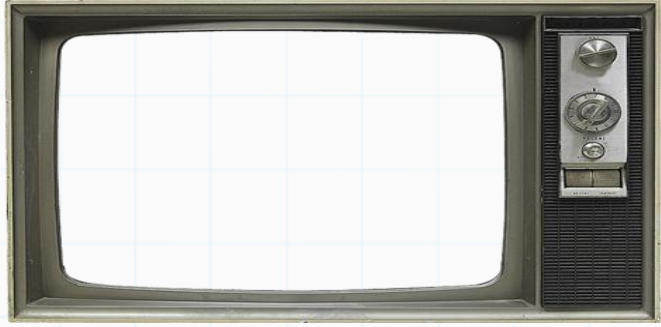


Comando SWITCH:
Apenas o bloco no qual a condição é verdadeira é executado




```
Portugol

...
Dada VARIAVEL
caso valor1 então
       INSTRUÇÃO 1
       ...
caso valor2 então
       INSTRUÇÃO 1
       ...
caso valor3 então
       INSTRUÇÃO 1
       ...
```

Decisão



Comando SWITCH:

Apenas o bloco no qual a condição é verdadeira é executado

Portugol

```
...
Dada VARIAVEL
caso valor1 então
      INSTRUÇÃO 1
      ...
caso valor2 então
      INSTRUÇÃO 1
      ...
caso valor3 então
      INSTRUÇÃO 1
      ...
```

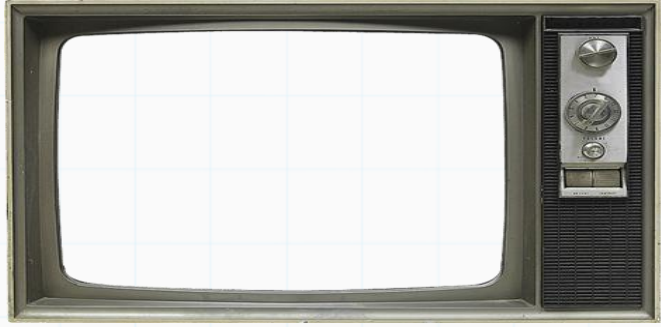
C

```
...
switch (VARIAVEL)
{
   case valor1:
        INSTRUÇÃO 1;
        ...
        break;
   case valor2:
        INSTRUÇÃO 2;
        ...
        break;
   case valor3:
        INSTRUÇÃO 3;
        ...
        break;
   default:
        INSTRUÇÃO 4;
        ...
        break;
}
```

- O comando `break` é importante para finalizar cada caso do switch
- A cláusula `default` é opcional. Ela é executada apenas quando nenhum dos casos combinar com a expressão
- No case, não é necessário usar chaves, mesmo que tenhamos mais de um comando
- o `break` indicará o final de cada caso

Se não for usado o `break` para finalizar um caso a execução prossegue para o próximo caso

Decisão

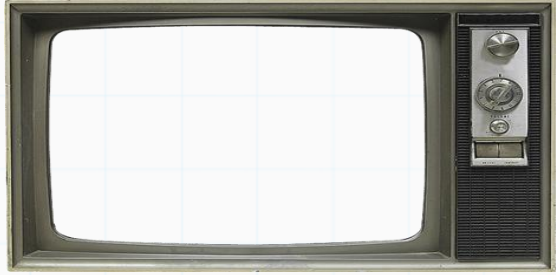


Comando SWITCH:

Apenas o bloco no qual a condição é verdadeira é executado

```
int mes;
...
switch (mes) {
        case  1:
              printf("JAN");
              break;
        case  2:
              printf("FEV");
              break;
        case  3:
              printf("MAR");
              break;
        ...
        case  12:
              printf("DEZ");
              break;
        default:
              printf("Mês Inválido");
              break;
}
```

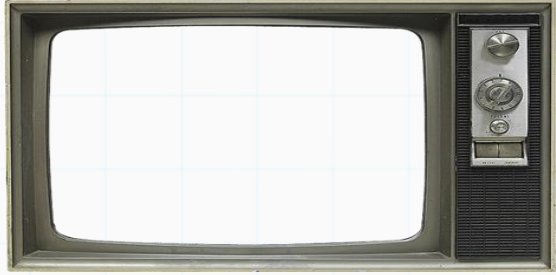
Decisão



Exemplo: O que será escrito ?

```c
#include <stdio.h>
void main()
{
    int ch=2;

    switch (ch)
    {
        case 1:
          printf("1\n");
          break;
          printf("hi");
       default:
          printf("2\n");
    }
}
```

Decisão

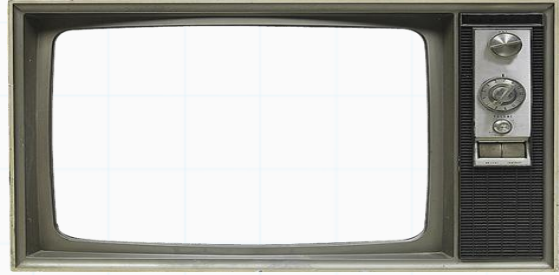


Exemplo: O que será escrito ?

```c
#include <stdio.h>
void main()
{
    int ch=2;

    switch (ch)
    {
        case 1:
          printf("1\n");
          break;
          printf("hi");
       default:
          printf("2\n");
    }
}
```

```c
#include <stdio.h>
void main()
  {
      int ch=1;

      switch (ch)
      {
          case 1:
             printf("1\n");
          default:
             printf("2\n");
      }
  }
```

Decisão



Exemplo: O que será escrito ?

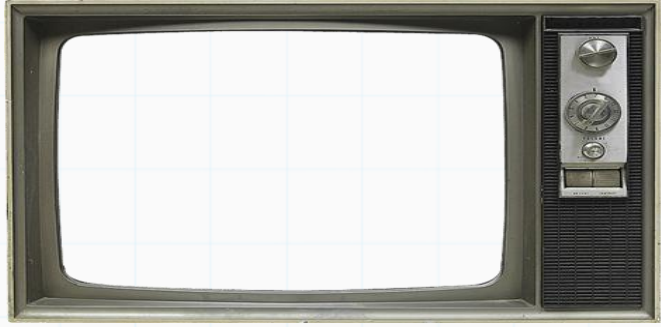
```c
#include <stdio.h>
void main()
{
    int ch=2;

    switch (ch)
    {
        case 1:
          printf("1\n");
          break;
          printf("hi");
       default:
          printf("2\n");
    }
}
```

2

```c
#include <stdio.h>
void main()
  {
      int ch=1;

      switch (ch)
      {
          case 1:
             printf("1\n");
          default:
             printf("2\n");
      }
  }
```

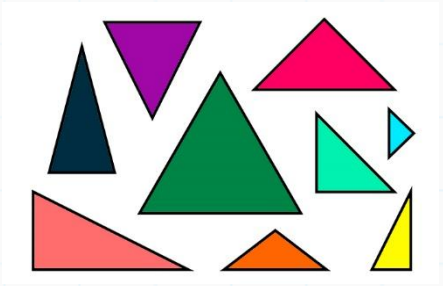
1
2

Decisão



Exercício 1) Triângulos: Faça um programa que leia o tamanho dos 3 lados de um triangulo (inteiro), e diga qual tipo ele é

Equilátero = todos os lados iguais
Isósceles=apenas dois lados iguais
Escaleno= nenhum lado igual



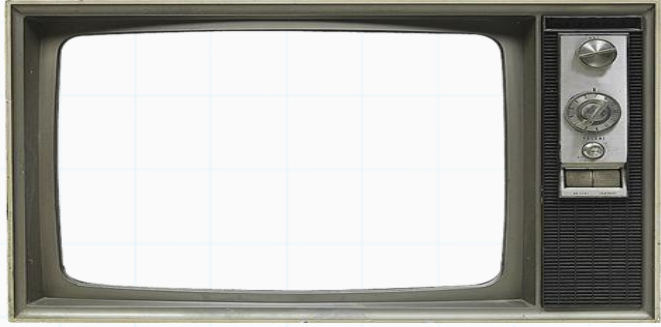
Esqueleto básico de todo código

```c
#include<stdio.h>

int main()
{
    /* código */
    return 0;
}
```

```c
if (CONDIÇÃO)
{
   INSTRUÇÃO 1;
}
else
{
   INSTRUÇÃO 1;
}

&& -> e lógico
|| -> ou lógico

printf("a = %d",a);
scanf("%d",&n);
```

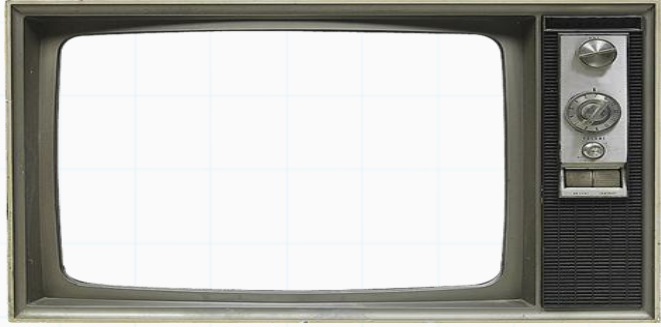
Use só o que aprendemos até hoje

Decisão



```c
#include <stdio.h>
int main()
{
    int side1, side2, side3;
    printf("3 lados: ");
    scanf("%d%d%d", &side1, &side2, &side3);

    if(side1==side2 && side2==side3)
    {
        printf("Equilatero.");
    }
    else if(side1==side2 || side1==side3 || side2==side3)
    {
        printf("Isosceles.");
    }
    else
    {
        printf("Scaleno.");
    }

    return 0;
}
```

Decisão



Exercício 2) Salário: Faça um programa que leia o número de horas trabalhadas (inteiro) na semana, e imprima o salário bruto (double), imposto pago (double) e salário líquido (double). Sabendo que:

Valor = 12 reais por hora
Extra (acima de 40 horas) = 1.5x o valor normal da hora

Imposto:

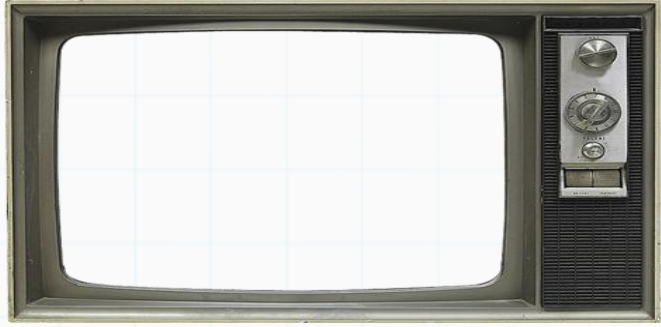
15% sobre os primeiros 300 reais
20% sobre os próximos 150 reais
25% sobre o resto


```c
#include<stdio.h>

int main()
{
    /* código */
    return 0;
}
```

Use só o que aprendemos até hoje

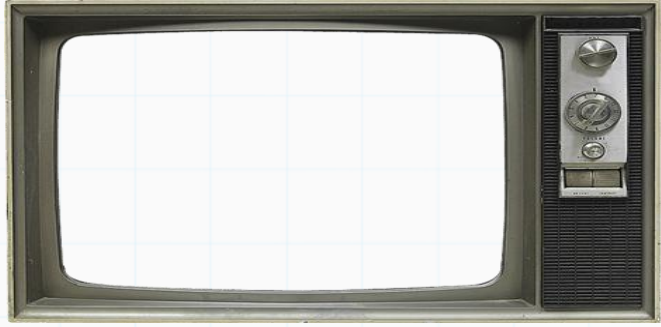
```c
if (CONDIÇÃO)
{
  INSTRUÇÃO 1;
}
else
{
  INSTRUÇÃO 1;
}

&& -> e lógico
|| -> ou lógico

printf("a = %d",a);
scanf("%d",&n);
```

Decisão

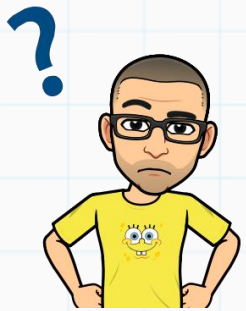


```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int horas = 0;
    double sb = 0.0;
    double imposto = 0.0;
    double sl = 0.0;

    printf("horas trabalhadas: ");
    scanf("%d", &horas);

    if (horas <= 40)
        sb = horas * 12;
    else
    {
      sb = 40 * 12;
      double extra = (horas - 40) * (12 * 1.5);
      sb += extra;
    }
    if (sb <= 300)
    {
        imposto = sb * 0.15;
    }
    else if(sb > 300 && sb <= 450)
    {
        imposto = 300 * 0.15;
        imposto += (sb - 300) * 0.20;
    }
    else if (sb > 450)
    {
        imposto = 300 * 0.15;
        imposto += 150 * 0.20;
        imposto += (sb - 450) * 0.25;
    }
    sl = sb - imposto;

    printf("salario bruto: %.2f\n", sb);
    printf("imposto: %.2f\n", imposto);
    printf("salario liquido: %.2f\n", sl);

    return 0;
}
```

Decisão



Exercício 3) Poker: Receba 3 valores inteiros que representam 3 cartas de baralho. Se os três números forem iguais, imprime a palavra “trinca". Se apenas dois deles forem iguais, imprima a palavra “par”. Se os 3 forem diferentes mas consecutivos imprima a palavra “sequencia”, senão forem consecutivos imprima a palavra “lascou”.



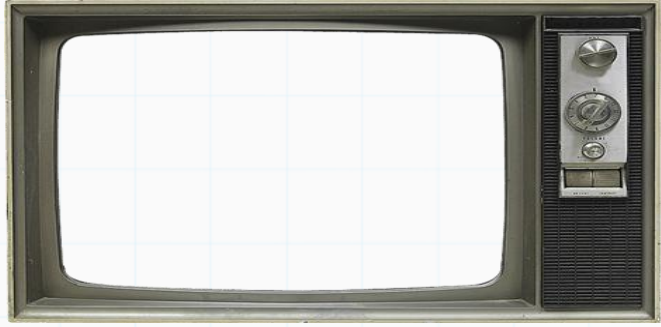
```
if (CONDIÇÃO)
{
   INSTRUÇÃO 1;
}
else
{
   INSTRUÇÃO 1;
}

&& -> e lógico
|| -> ou lógico

printf("a = %d",a);
scanf("%d",&n);
```

Lembre-se, uma sequencia pode ser:

- 3,4,5
- 5,4,3
- 4,3,5
- ...

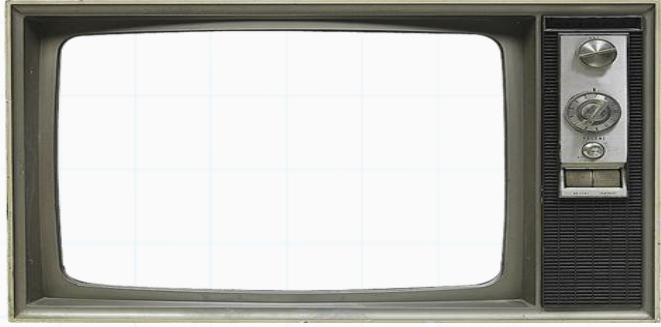
Use só o que aprendemos até hoje

Decisão



```c
#include <stdio.h>
int main() {
    int a,b,c;

    printf("a=");
    scanf("%d", &a);
    printf("b=");
    scanf("%d", &b);
    printf("c=");
    scanf("%d", &c);

    if (a==b && b==c)
    {
        printf("trinca");
    }
    else
    {
        if ((a==b) || (a==c) || (b==c))
        {
            printf("par");
        }
        else
        {
            if ((a==b+1) && (a==c+2) || (a==c+1) && (a==b+2) ||
                (b==a+1) && (b==c+2) || (b==c+1) && (b==a+2) ||
                (c==a+1) && (c==b+2) || (c==b+1) && (c==a+2))
                    printf("sequencia");
            else
                printf("lascou");
        }
    }
    return 0;
}
```

Decisão

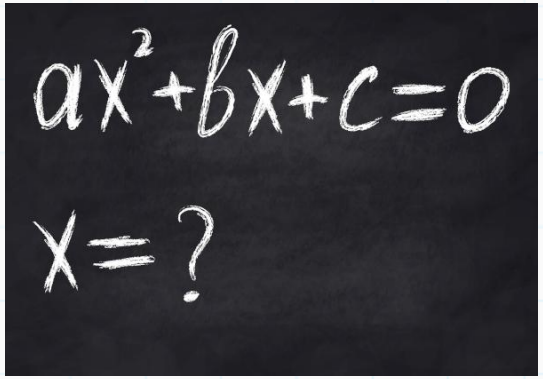


Exercício 4) Raízes: Dado uma equação do segundo grau $ax^2 + bx + c$, imprima suas raízes sabendo que:

descriminante = $b^2 – 4ac$

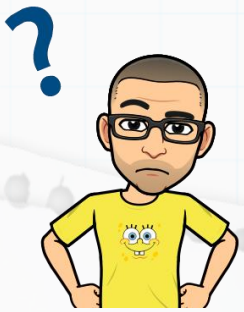
Caso descriminante = 0
raiz = -b/2a
Caso descriminante > 0
raiz1 = (-b+ raizq(descriminante)) / 2a
raiz2 = (-b- raizq(descriminante)) / 2a

Caso descriminante < 0
escreva “sem raízes reais”




```
switch (CONDIÇÃO)
{
  case valor1:
        INSTRUÇÃO 1;
        break;
  case valor2:
        INSTRUÇÃO 2;
        break;
default:
        INSTRUÇÃO 3;
        break;
}

&& -> e lógico
|| -> ou lógico

printf(“a = %d”,a);
scanf(“%d”,&n);
```

Porem, a única estrutura de decisão que você pode usar é o switch !

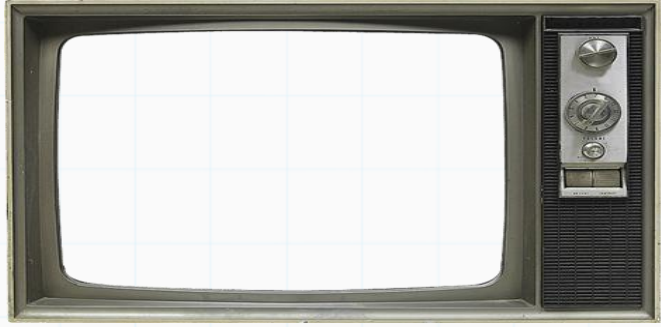


```
#include <math.h>
x=16
y=sqrt(x)
```

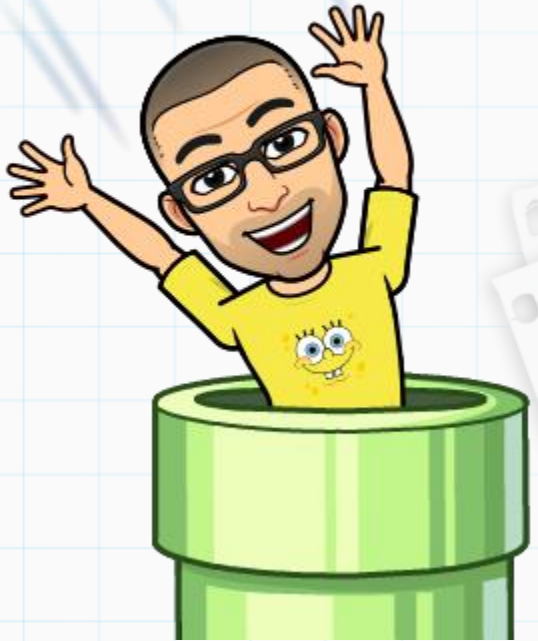
Raiz quadrada

Use só o que aprendemos até hoje

Programação Estruturada



Até a próxima



Slides baseados no curso de Aline Nascimento