

LP - Lógica de Programação

Conteúdo

1	Introdução à lógica de programação	2
2	Representações Visuais de algoritmo	5
2.1	Scratch	5
3	Portugol	7
3.1	Estrutura básica	7
3.2	Leitura e escrita	7
3.3	Processamento de dados	8
3.4	Outros tipos de variáveis	9
3.5	Exercícios	9
4	Expressões e operadores	9
4.1	Trabalho	13
4.2	Exercícios	13
5	Teste de mesa	14
6	Linguagem C	14
6.1	Tipos de dados	15
6.2	Leitura e impressão	15
6.3	Exercícios	15
7	Estrutura condicional simples e composta	16
7.1	If ou If...Else	16
7.1.1	Exercícios:	17
7.2	Switch Case	17
8	Estrutura de repetição	18
8.1	FOR	18
8.2	While e Do While	19
9	Torres de Hanói	19
10	Boas práticas de programação	21
10.1	Código DRY	22

1 Introdução à lógica de programação

Lógica:

- organização e explicação de um pensamento.
- analisar argumentos em uma sequência lógica, chegando a uma conclusão.

Exemplos:

$$\begin{cases} \text{Todo mamífero bebe leite} \\ \text{O homem bebe leite} \end{cases} \Rightarrow \text{O homem é mamífero}$$

Este exemplo apresenta argumentos fracos, pois, por exemplo, uma pessoa poderia ter uma ave de estimação que bebesse leite. Um argumento mais forte seria, ao invés de usar o termo 'bebe', usássemos o termo 'produz', pois esta é uma característica única aos mamíferos. Ficando assim:

$$\begin{cases} \text{Todo mamífero produz leite} \\ \text{O homem (ser humano) produz leite} \end{cases} \Rightarrow \text{O homem (ser humano) é mamífero}$$

Devemos ter cuidado ao organizar os argumentos para não obtermos conclusões equivocadas, como no exemplo a seguir:

$$\begin{cases} \text{O pinguim é preto e branco} \\ \text{Alguns filmes antigos são em preto e branco} \end{cases} \Rightarrow \text{Alguns pinguins são filmes antigos}$$

Linguagem de programação: meio de comunicação entre computadores e humanos.

- Cria-se um arquivo de texto contendo a lógica do que deve ser feito
- arquivo é chamado de **programa**
- cada palavra de ordem é chamada **instrução**
- este arquivo é traduzido para linguagem entendida pelo computador → **executável** é gerado e interpretado diretamente pelo computador.

Algoritmo: é uma sequência lógica de passos que levam a um determinado objetivo. Podendo haver mais de um algoritmo para resolver um mesmo problema.

Exemplos:

- Vir de casa para a escola: a pé, de carona, de bicicleta.
- Rotina ao acordar: tomar café → escovar os dentes → trocar de roupa OU trocar de roupa → tomar café → escovar os dentes

Critério de escolha: algoritmo que melhor se adequar as nossas atividades. Alguns fatores: tempo, gasto financeiro (caso ir de carro ou de uber), gasto de memória (para programas de computador), entre outros.

Estrutura do Algoritmo

Início

Entrada: dados que devem ser lidos ou fornecidos pelo usuário

Processamento: onde acontecem os cálculos em si, os dados de entrada são manipulados a fim de se gerar uma resposta (um resultado)

Saída: são os resultados de seu processamento

Fim

Atividade: escreva um algoritmo para a resolução de um problema cotidiano, supondo que seu interlocutor é outro ser humano. Exemplos: fazer uma garrafa de café, percurso de sua casa até a escola a pé, como fritar um ovo, entre outros.

Abaixo a atividade adaptada de ordenação de um algoritmo. Usar esta antes da anterior, para melhor entendimento da turma.

Atividade adaptada: ordene os seguintes passos a fim de se ter um algoritmo de como preparar uma garrafa de café, supondo que seu interlocutor é outro ser humano.

Coloque o bule no fogão

Fim

Início

Encha um bule com 1l de água

Acenda o fogo da boca do fogão

Coloque o pó no coador

Coloque o papel no coador

Esvazie e lave a garrafa térmica

Encaixe o coador na garrafa térmica

Quando a água ferver, passe-a pelo coador

2 Representações Visuais de algoritmo

2.1 Scratch

O Scratch é uma plataforma de programação visual desenvolvida pelo MIT (Instituto de Tecnologia de Massachusetts) que foi criada com o objetivo de tornar a programação acessível e divertida para pessoas de todas as idades.

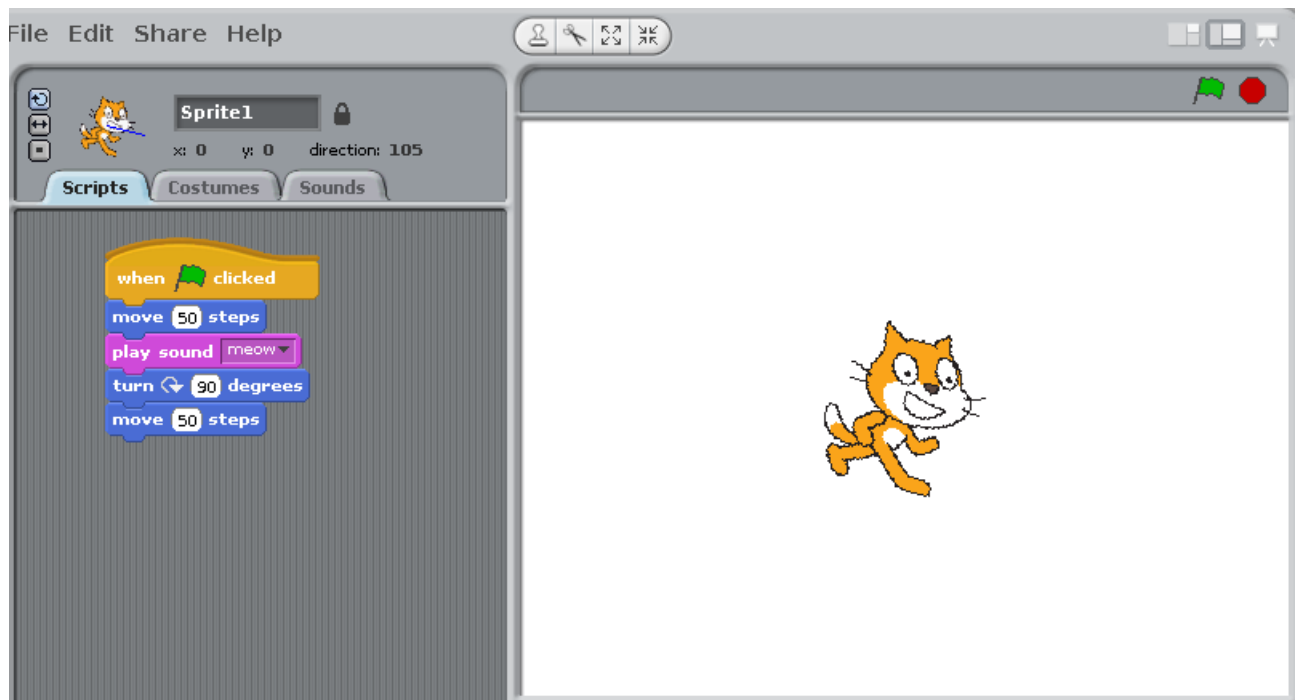
O Scratch é considerado mais acessível que linguagens de programação textuais, por se utilizar de uma interface gráfica que permite que programas sejam construídos com blocos encaixados, lembrando o brinquedo Lego. Utiliza uma sintaxe comum a muitas linguagens de programação. É diferente de outras linguagens, não tem nenhum tipo de pontuação obscura.

Cada bloco da linguagem contém um comando em separado, que podem ser agrupados livremente caso se encaixem. E os comandos podem ser modificados através de menus barra de snirks.

Mostrar todas as opções no menu do Scratch

Exercício: faça um programa no *Scratch* seguindo o seguinte algoritmo:

- Quando *start*
- Mova 50 passos
- Faça o som "Miau"
- Vire 90° no sentido horário
- Mova 50 passos



Exercícios:

1. Faça um programa no *Scratch* seguindo o seguinte algoritmo:
 - Quando *start*

- Ande 80 passos
- Gire 90° no sentido anti-horário
- Diga "Boa tarde, professora" por 2 segundos
- Pense "Quero ir embora" por 2 segundos

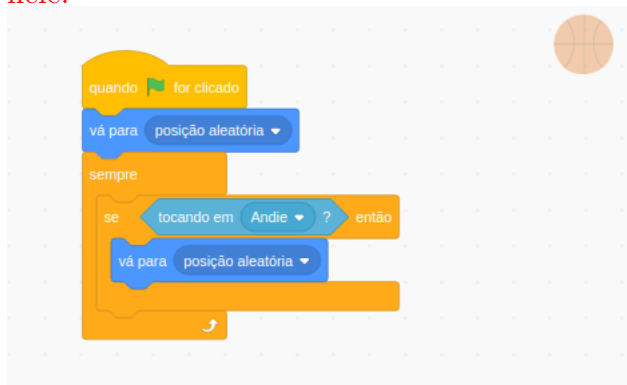
2. Faça um programa no *Scratch* seguindo o seguinte algoritmo:

- Quando a tecla *espaço* for clicada
- Volte à posição inicial (0,0)
- Fique na vertical

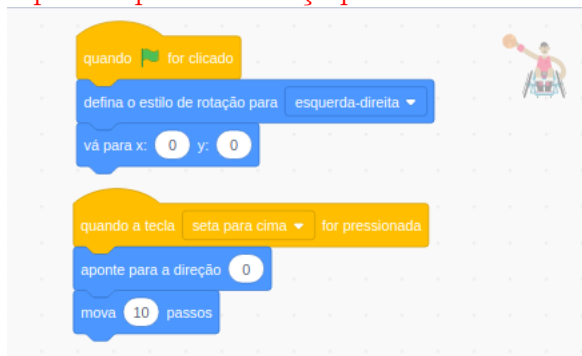
Trabalho 1

Faça um jogo em **Scratch** em que o ator deve se movimentar na tela e capturar um objeto que aparece em posições aleatórias.

Código para fazer o objeto ir para posição aleatória quando o ator (Andie) encosta nele:



Código para fazer o ator (Andie) se movimentar de acordo com as setas e para que ele fique sempre 'de cabeça para cima'.



Abaixo uma tabela com quais ângulos devemos colocar no comando 'aponte para a direção'.

Direção	Ângulo
Para cima	0
para baixo	180
Para a direita	90
Para a esquerda	-90 ou 270

Trabalho 2

A partir das ferramentas aprendidas no Trabalho 1, faça um jogo (tema livre).

Grupos: de até 5 alunos, mas pode ser feito individual.

Tempo: 2 semanas (4 aulas)

Avaliação por pares: os colegas devem testar (jogar) os jogos dos colegas, dar algum *feedback* para alguma melhoria que achar necessária.

3 Portugol

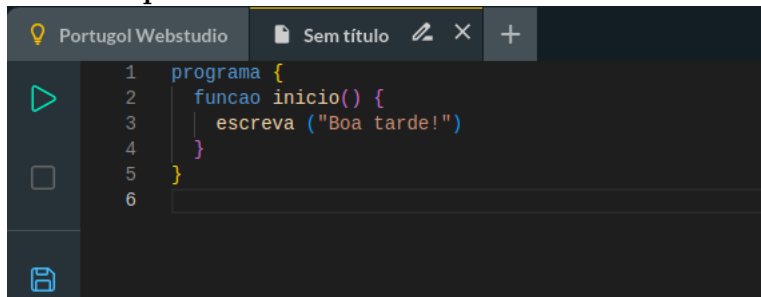
Portugol, também conhecido como Português estruturado, é uma família de linguagens de programação que possui como base a língua portuguesa. Algumas de suas variações podem ser consideradas pseudocódigo, e outras são linguagens completas, livres de contexto, com gramáticas definidas e implementações em editores ou compiladores. Usaremos a versão online disponível em <https://dgadelha.github.io/Portugol-Webstudio/>.

3.1 Estrutura básica

Em portugol temos a seguinte estrutura básica:

- **Primeira linha:** `programa`. Inicia-se o código digitando a palavra `programa`.
- **Corpo do programa:** `entre { }`. Tudo que o programa deve fazer deve ser escrito entre chaves.
- **A função principal:** `funcao inicio ()`. Também seguida de `entre { }`.

Exemplo: Boa tarde



```
1 programa {
2     funcao inicio() {
3         escreva ("Boa tarde!")
4     }
5 }
6
```

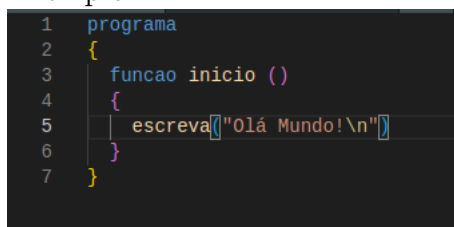
3.2 Leitura e escrita

A escrita é feita da forma:

`escreva ("Texto a ser impresso na tela")`

Para pular uma linha, usamos o comando `\n` dentro das aspas do texto a ser impresso.

Exemplo:



```
1 programa
2 {
3     funcao inicio ()
4     {
5         escreva("Olá Mundo!\n")
6     }
7 }
```

Portugol é uma linguagem de declaração (declarativa), ou seja, as variáveis devem ser declaradas antes de serem utilizadas. A declaração das variáveis é feita da seguinte forma:

inteiro **variavel**

real **variavel**

A leitura de variáveis é feita da seguinte forma:

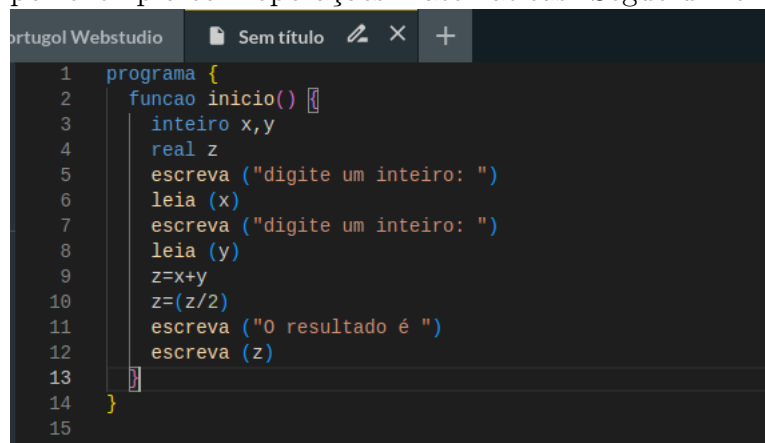
leia (**variável**)

Exercícios

1. Faça um programa em Portugol que imprime na tela os seguintes textos:
 - Boa tarde, professora.
 - Meu nome é **Seu nome**
 - Hoje é dia **Data do dia**
 - Boa tarde!
Hoje está um dia muito bonito!
Gostaria de estar na praia.
2. Faça um programa em Portugol que lê um inteiro e imprime na tela: O número digitado foi: **número**
3. Faça um programa em Portugol que lê dois inteiros e imprime na tela: Os números digitados foram: **número1** e **número2**

3.3 Processamento de dados

Após a leitura (comandos de entrada) dos dados feita, pode-se processar esses dados; por exemplo com operações matemáticas. Segue um exemplo:



```
1 programa {
2   funcao inicio()
3     inteiro x,y
4     real z
5     escreva ("digite um inteiro: ")
6     leia (x)
7     escreva ("digite um inteiro: ")
8     leia (y)
9     z=x+y
10    z=(z/2)
11    escreva ("O resultado é ")
12    escreva (z)
13
14 }
15
```

Operação	Símbolo
Soma	+
Subtração	-
Multiplicação	*
Divisão	/

Lembrando que o computador obedece à ordem de precedência das operações matemáticas, a divisão e a multiplicação são feitas antes da soma e da subtração, para efetuar uma soma/subtração antes de uma divisão/multiplicação devemos usar parênteses.

3.4 Outros tipos de variáveis

Temos as variáveis do tipo *string*, ou seja, do tipo texto.

Tipo	Descrição
caracter	apenas um símbolo (ex: uma letra)
cadeia	um texto até que o usuário digite enter

3.5 Exercícios

1. Faça um programa em Portugol que lê 2 números inteiros e calcula a média entre eles.
2. Faça um programa em Portugol que lê 3 números inteiros e calcula a média entre eles.
3. **[VALE VISTO]** Faça um programa em Portugol que lê 2 números reais e:
 - a) Soma os dois
 - b) Subtrai o segundo do primeiro
 - c) Multiplica os dois
 - d) Divide o primeiro pelo segundo
4. Faça um programa em Portugol que lê 4 números reais e calcula a média entre eles.
5. **[VALE VISTO]** Faça um programa que lê um caracter x e uma cadeia y, e imprime: "Texto digitado em x: x e texto digitado em y: y". Faça testes digitando o mesmo texto nas duas entradas. O que você notou?
6. Faça um programa em portugol que lê uma *string* com o nome do usuário, lê duas notas de 0 a 10 e calcula a média dessas notas e imprime a seguinte mensagem: "<nome> sua média final foi <media>"
7. **[VALE VISTO]** Faça um programa em portugol que lê duas *strings* com o nome de dois usuários, lê duas notas de 0 a 10 para cada, na seguinte ordem: nome1, nota1₁ e nota1₂ e nome2, nota2₁ e nota2₂ e calcula a média dessas notas de cada usuário e imprime a seguinte mensagem: "<nome1> sua média final foi <media1> e <nome2> sua média final foi <media2>"

4 Expressões e operadores

Expressão lógica: também chamada de *proposição*. É uma expressão algébrica.

Operadores lógicos: E, OU, SE...ENTÃO, SE e SOMENTE SE, NÃO, XOR (OU exclusivo), NOR, NAND...

Os operadores lógicos relacionam duas proposições. Aparecem entre duas proposições.

Valor lógico: resultado de operadores lógicos sobre proposições. Apresenta dois estados: Verdadeiro ou Falso.

Obs: Tautologia: verdadeira para todas as possíveis entradas. É sempre verdadeira.

Tabela Verdade: auxilia na definição do valor lógico de uma proposição, avaliando seus operadores lógicos.

E (AND), símbolo \wedge

é verdade quando todas as entradas são verdade

Tabela verdade:

p	q	p \wedge q
F	F	F
F	V	F
V	F	F
V	V	V

Exemplo: Bruna fez um pedido na lanchonete, ela queria um refrigerante E uma batata frita.

Refrigerante	Batata Frita	Acertou o pedido?
Não trouxe refrigerante	Não trouxe batata frita	Não
Não trouxe refrigerante	Trouxe batata frita	Não
Trouxe refrigerante	Não trouxe batata frita	Não
Trouxe refrigerante	Trouxe batata frita	Sim

OU (OR), símbolo \vee

é verdade quando ao menos uma das entradas é verdade

Tabela verdade:

p	q	p \vee q
F	F	F
F	V	V
V	F	V
V	V	V

Exemplo: Bruna fez um pedido na lanchonete, ela queria um sorvete de chocolate OU de baunilha.

NÃO (NOT), símbolo: \neg (ou \sim em algumas bibliografias - menos usado) A proposição no caso terá o valor invertido, quando **verdade** passará para **falso** e vice-versa.

Tabela verdade:

Sorvete de chocolate	Sorvete de baunilha	Acertou o pedido?
Não trouxe sorvete de chocolate	Não trouxe sorvete de baunilha	Não
Não trouxe sorvete de chocolate	Trouxe sorvete de baunilha	Sim
Trouxe sorvete de chocolate	Não trouxe sorvete de baunilha	Sim
Trouxe sorvete de chocolate	Trouxe sorvete de baunilha	Sim

p	$\neg p$
F	V
V	F

Observações: o **NÃO** funciona como um interruptor, ele altera o estado das proposições que o seguem. Alguns exemplos não são intuitivos, principalmente quando temos duas negações na mesma frase, o que para o português falado parece que a frase está com conotação negativa, porém para a lógica ela terá uma conotação positiva.

Exemplo: quando falamos "Não tem ninguém em casa" temos o entendimento que a casa está vazia. Note que para a lógica é justamente o contrário.

SE...ENTÃO, símbolo \rightarrow

é falso quando a variável que antecede for **verdade** e a seguinte for **falsa**.

Tabela verdade:

p	q	$p \rightarrow q$
F	F	V
F	V	V
V	F	F
V	V	V

Exemplo: Amanhã, se fizer sol, então eu irei à praia.

- Fez sol e eu não estou na praia: Falso
- Fez sol e eu estou na praia: Verdade
- Não fez sol: os dois casos são Verdade, pois eu posso tanto ter ido ou não ter ido à praia.

Exercício: nas seguintes proposições, escreva os casos que seu valor lógico é **Falso**:

- Se este animal é um papagaio, então ele é um pássaro.
- Se amanhã chover, então eu irei ao cinema.
- Se esta bicicleta é vermelha, então aquele livro é verde.
- Se você tomar sorvete, então você ficará resfriado.
- Se eu passar de ano, então eu irei viajar para a Disney.

f) Se esta caixa estiver vazia, então vamos enchê-la de coisas.

Contrapositiva: quando temos uma proposição do tipo "Se **p** então **q**" temos a contrapositiva "Se **não q** então **não p**". As duas são equivalentes, ou seja, possuem o mesmo resultado na tabela verdade.

p	q	$\neg q$	$\neg p$	$\neg q \rightarrow \neg p$
F	F	V	V	V
F	V	F	V	V
V	F	V	F	F
V	V	F	F	V

Equivalente: outra equivalente para a proposição "Se...então" é " $\neg p \vee q$ " (não p ou q). Por exemplo: a proposição "Se esta caixa é verde, então o ventilador é amarelo" é equivalente a "A caixa não é verde ou o ventilador é amarelo". Isso pode ser comprovado fazendo a tabela verdade de cada uma e comparando os resultados:

p	q	$\neg p$	$\neg p \vee q$
F	F	V	V
F	V	V	V
V	F	F	F
V	V	F	V

Obs: apesar de parecerem bem diferentes as frases, de terem significado diferente, para a lógica matemática elas são equivalentes. Na dúvida, devemos fazer a tabela verdade de cada proposição e comparar os resultados.

4.1 Trabalho

1. Qual critério devemos avaliar para que duas proposições sejam equivalentes?
2. Qual das proposições abaixo é uma tautologia?

a) $(p \vee q) \vee \neg q$

c) $(p \wedge q) \vee \neg q$

b) $(p \vee q) \wedge \neg q$

d) $(p \wedge q) \wedge \neg q$

3. Escreva um algoritmo para a resolução do seguinte problema: Marcos viaja entre duas cidades A e B, a distância entre elas é S. O primeiro terço da viagem ele percorre a uma velocidade média de x, os outros dois terços a uma velocidade média y. As entradas do problema são S, x e y. Qual a velocidade média do percurso total?

4.2 Exercícios

1. Faça a tabela verdade das seguintes proposições:

a) $p \vee q$

d) $p \wedge \neg q$

b) $p \wedge q$

e) $\neg p \vee q$

c) $p \vee \neg q$

f) $\neg p \wedge q$

2. As bibliotecas utilizadas na linguagem C são de qual tipo de arquivo? (Qual a extensão usada no arquivo)
3. Qual função (comando) é usada para se imprimir texto em C? Como fazemos para ao final do texto a ser impresso, também se pule uma linha?
4. Qual caractere é usado em C para indicar fim de linha em um programa?
5. De quais formas podemos escrever comentários no corpo do programa?
6. Para você, qual a importância dos comentários colocados ao longo do programa?

5 Teste de mesa

Teste de mesa é uma simulação da execução de um programa de forma manual, geralmente feita no papel. Não há regras rígidas para criar um teste de mesa, mas geralmente ele é feito de uma de duas formas.

É um processo manual que é utilizado para validar a lógica de um determinado algoritmo. Ele é utilizado principalmente em algoritmos quando a linguagem utilizada não possui nenhuma ferramenta automatizada de depuração. Como as linguagens de programação costumam possuir tais ferramentas, é mais comum utilizá-las a fazer o teste de mesa propriamente dito, embora para quem ainda é iniciante, eu particularmente ainda recomendo utilizá-lo, visto que provavelmente não terá domínio sobre a ferramenta de depuração.

6 Linguagem C

A linguagem C é uma linguagem de programação de alto nível que foi criada nos anos 1970 para desenvolvimento de sistemas operacionais. Ela é amplamente utilizada em sistemas embarcados.

Algumas observações:

- Os arquivos de programa em C devem ser salvos com a extensão `.c`.
- Deve-se incluir as bibliotecas (arquivos do tipo `.h` no início do programa). A biblioteca `stdio.h` deve ser incluída em todos os programas, pois é a biblioteca responsável pelas funções de leitura e escrita, por exemplo.
- É uma linguagem de programação de declaração, ou seja, as variáveis devem ser declaradas (tipo e nome) antes de serem utilizadas no corpo do programa.
- Cada linha de comando deve ser finalizada pelo caractere `;`.
- O programa principal é chamado de `main` (se declarada como `int main()` deve-se finalizar o programa com um `return 0;`).
- Para pular uma linha, deve-se imprimir um `'\n'`
- Comentários no programa podem ser colocados após `/**` (comenta apenas a linha) ou entre `/*` e `*/`. Exemplo:

```
#include <stdio.h>

int main(void) {
    float x,y,m;
    //isso é um comentário
    /*Isso
    também
    é um
    comentário*/
    scanf("%f", &x);
    scanf("%f", &y);
    m=y;
    if (x>y)
        {m=x;}
    printf("O resultado é: %f\n",m );
    printf("%f %f",x ,y);

    return 0;
}
```

- Não suporta Classes e Objetos (C++).

Exemplo "Hello World":

```
#include <stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}
```

6.1 Tipos de dados

Segue a tabela com os tipos mais usados de dados em C e seu tamanho, descrição e qual especificador deve ser usado:

Tipo	Tamanho	Descrição	Especificador
int	2 ou 4 bytes	números inteiros, sem casa decimal	%d ou %i
float	4 bytes	números reais, suporta até 6-7 dígitos	%f ou %F
double	8 bytes	números reais, suporta até 15 dígitos	%lf
char	1 byte	1 único caractere ou valor ASCII	%c
string	509 caracteres	texto	%s
booleano	1 bit	1 ou 0, true ou false	%d

Obs: o tipo *booleano* precisa da inclusão de uma biblioteca específica `stdbool.h`.

6.2 Leitura e impressão

Para leitura de dados usa-se o comando `scanf` seguido de `%x`, onde x (especificador de formato) varia conforme o tipo de dados que está sendo lido, essa leitura deve ser feita em `&y`, onde y é o nome da variável.

Exemplo:

```
float x,y;
scanf("%f", &x);
scanf("%f", &y);
```

Para impressão também é necessário indicar o especificador `%x`, mas a variável é passada diretamente.

Exemplo:

```
printf("O resultado é: %f\n",m );
printf("%f %f",x ,y);
```

6.3 Exercícios

1. Faça um programa que lê dois inteiros, soma e imprime o resultado.
2. Faça um programa que lê dois inteiros, faz o produto e imprime o resultado.
3. Faça um programa que lê três números reais, soma e imprime o resultado..
4. Faça um programa que lê três números reais, faz o produto e imprime o resultado.

5. **EXTRA** Pedro viaja de carro da cidade A para a cidade B e retorna imediatamente. Na ida ele viaja a uma velocidade média de V_a e na volta ele viaja a uma velocidade média de V_b . Faça um programa que recebe dois valores reais para V_a e V_b e retorna V_m , a velocidade média total da viagem.

7 Estrutura condicional simples e composta

7.1 If ou If...Else

Condições do tipo **Se** ou **Se...Então**. A linguagem C suporta condições lógicas. Segue a tabela com a sintaxe específica para cada caso:

Caso	Sintaxe
a menor que b	$a < b$
a menor ou igual a b	$a \leq b$
a maior que b	$a > b$
a maior ou igual a b	$a \geq b$
a igual a b	$a == b$
a diferente de b	$a != b$

A sintaxe do condicional é do tipo:

```
#include <stdio.h>

int main() {
    if (condição){
        //operações caso condição é verdade
    }
    return 0;
}
```

Quando usamos o **IF...ELSE**, teremos uma condição, mas duas operações distintas, uma quando a condição for verdade e a segunda quando a mesma for falsa. Segue o exemplo:

```
#include <stdio.h>

int main() {
    if (condição){
        //operações caso condição1 seja verdade
    }
    else {
        //operações caso condição1 seja falsa
    }
    return 0;
}
```

A condição do **IF** deve vir entre parênteses e essa estrutura suporta mais de uma condição. Neste caso, elas devem estar ligadas por conectivos lógicos **E** ou **OU**, segue a sintaxe:

Caso	Sintaxe
E	$\&\&$
OU	$\ \ $

Segue exemplo:


```
#include <stdio.h>

int main() {
    if ((condição1)&&(condicao2)){
        //operações caso condição1 E condicao2 são verdade
    }

    if ((condição3)|| (condicao4)){
        //operações caso condição3 OU condicao4 são verdade
    }
    return 0;
}
```

7.1.1 Exercícios:

1. Faça um programa que lê dois números inteiros e retorna o maior entre eles.
2. Faça um programa que lê três números reais e retorna o menor entre eles.
3. Faça um programa que lê quatro números reais, caso o terceiro número seja maior que o primeiro, ele faz o produto do segundo pelo quarto e imprime: "Produto: x", onde x é o resultado; caso contrário ele faz a soma do segundo com o quarto e imprime: "Soma: x", onde x é o resultado.
4. **EXTRA** Faça um programa que lê 4 números reais (a, b, c, d) e caso $a = d$ E b seja diferente de c imprima 'Deu certo!!!'; caso contrário, se $b = c$ OU a seja diferente de d imprima 'Esse não!!'; caso contrário imprima 'Eita'. Quais casos o programa irá imprimir 'Eita'.

7.2 Switch Case

Quando temos várias condicionais a serem feitas, podemos usar a estrutura **Switch**, esta analisa uma expressão e suas possíveis opções de resultado, os *cases*, cada caso terá um código a ser executado, seguido de um *break*; e ao final devemos ter um caso *default*, que é quando nenhum *case* foi selecionado (não precisa de *break*;). Segue a sintaxe:

```
switch (expressao){
    case x:
        //codigo a ser executado caso o valor da expressão seja x
        break;
    case y:
        //codigo a ser executado caso o valor da expressão seja y
        break;
    default:
        //codigo a ser executado caso o valor da expressão não seja nem x nem y
}
```

Observações:

- os valores dos cases não podem ser repetidos
- os valores dos cases não podem ser variáveis
- o valor de cada case deve ser do mesmo tipo da expressão switch
- a variável passada no switch deve ser de um dos tipos aceitos pela estrutura (short, byte, long, int, enum, string)

- Comparativo com *IF...ELSE*

- **velocidade:** dependendo da quantidade de casos que precisam ser testados no código, a velocidade de execução pode ser afetada. Por isso, se o número de casos for maior que 5, é interessante considerar utilizar a estrutura switch e não o if else
- **valores fixos vs valores booleanos:** o if else é melhor para testar condições booleanas, já a estrutura switch é melhor para testar valores fixos
- **legibilidade do código:** em muitos casos, o switch case ajuda a manter o código limpo, pois evita que uma quantidade muito grande de if else if sejam usados no programa
- **expressão de teste:** outro ponto importante que deve ser analisado é a expressão que será usada como base para o teste condicional. Isso porque o switch case aceita apenas valores fixos para teste. Por outro lado, o if else também é capaz de testar faixas de valores e condições além de valores fixos

Segue um exemplo que testa se número é par ou ímpar:

```
#include <stdio.h>
int main(void) {
    int x;
    scanf("%d", &x);
    x=x%2;
    switch (x){
        case 0:
            printf("par\n");
            break;
        case 1:
            printf("ímpar\n");
            break;
        case -1:
            printf("ímpar negativo\n");
            break;
        default:
            printf("? \n");
    }
    return 0;
}
```

8 Estrutura de repetição

8.1 FOR

- Laço de repetição de quantidade finita
- Possui um início, um fim e um incremento fixos e pré-programados
- Utiliza variáveis auxiliares do tipo inteiro, geralmente se usa: i, j, k

Sintaxe:

```
for(início; fim; incremento){
```

```
}
```

Segue o problema de, ler um número inteiro x e calcular o produto de todos os inteiros de 1 a x (calcular o fatorial). Não sabemos quantas multiplicações devem ser feitas se o valor de n será passado pelo usuário. Dessa forma, podemos usar o laço de repetição **for** sobre uma variável auxiliar i variando de 1 até n , ou seja, **início: $i=1$** e **fim: $i \leq x$** como incremento de 1 em 1 **incremento: $i=i+1$ ou $i++$** (mais usado).

```
#include <stdio.h>
int main(void) {
    int x,i,f;
    scanf("%d", &x);
    f=1;
    for (i=1;i<=x;i++){
        f=f*i;
    }
    printf("O valor do fatorial é %d\n",f);
    return 0;
}
```

8.2 While e Do While

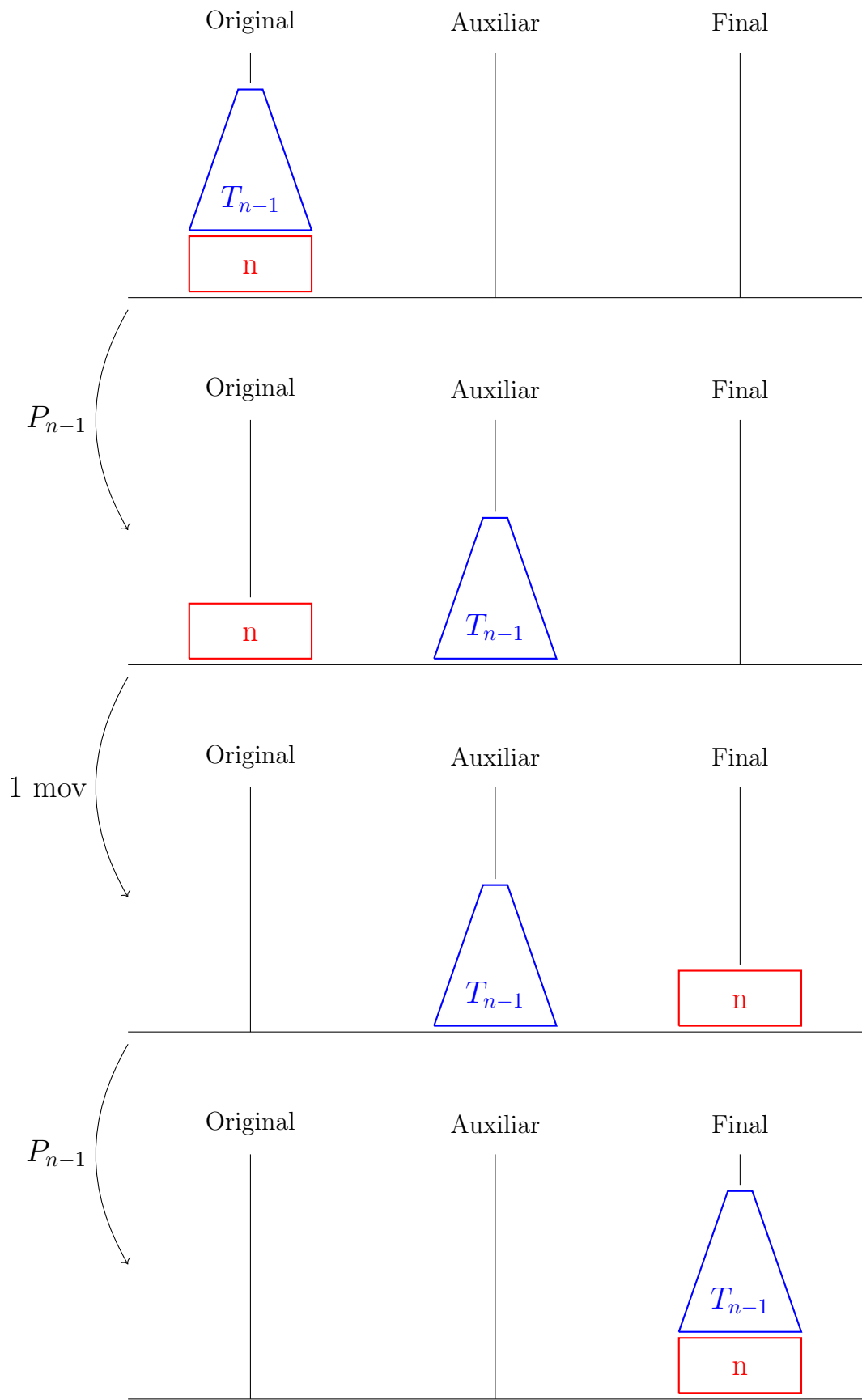
9 Torres de Hanói

O problema das Torres de Hanói consiste em se mover uma torre de peças de tamanhos diferentes, com peças maiores embaixo das menores, de uma haste de uma extremidade do jogo para a haste da outra extremidade com as seguintes regras:

- Deve-se mover apenas uma peça de cada vez
- Não se pode colocar uma peça maior sobre uma peça menor

A ideia do jogo é contar quantos passos (movimentos) a pessoa gasta para solucionar o problema. Considere:

- n : numeração da peça, por exemplo, a primeira peça é a peça 1, a oitava peça é a peça 8, a enésima peça é a peça n .
- T_n : a torre formada por n peças
- P_n : a quantidade de passos (movimentos) gastos para se resolver a torre T_n .



Para resolver a torre de tamanho n (T_n), resolve-se a torre anterior (T_{n-1}) na haste auxiliar, com mais um movimento, move-se a peça n para a haste final e por fim resolve-se a torre anterior (T_{n-1}) na haste final. Assim, temos a quantidade de passos para resolver

a torre T_n com a seguinte relação de recorrência:

$$P_n = 2 \cdot P_{n-1} + 1$$

Partindo do caso trivial, $n=1$, que é resolvido em 1 movimento, ou seja, $P_1 = 1$, é possível calcular P_2 com $P_2 = 2 \cdot P_1 + 1 = 2 \cdot 1 + 1 = 2 + 1 = 3$ e assim por diante, obtendo:

n	P_n
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255

Em sala: plotar os pontos no Geogebra e comparar com funções conhecidas: afim, segundo grau, exponencial, logaritmo. A fim de se observar que a função é uma exponencial de base 2.

Observe que os pontos se comportam como uma exponencial de base 2, comparando, temos:

n	P_n	2^n	$2^n - 1$
1	1	2	1
2	3	4	3
3	7	8	7
4	15	16	15
5	31	32	31
6	63	64	63
7	127	128	127
8	255	256	255

Com isso chegamos na fórmula fechada para a quantidade ótima de passos para a solução das Torres de Hanói:

$$P_n = 2^n - 1$$

10 Boas práticas de programação

São técnicas que visam melhorar a qualidade do código, tornando-o mais legível, simples e eficiente. Algumas dessas práticas são:

- **Nomes descritivos:** Escolher nomes significativos para variáveis e funções (ou métodos), que representem bem o que elas significam. Estes nomes de variáveis não precisam ser apenas uma letra.
- **Convenções de nomenclatura:** Seguir as convenções de nomenclatura da linguagem de programação que se está trabalhando.

- **Formatação e indentação:** Cuidar da formatação e da indentação do código. Usar sempre pular linhas e espaços (tab) para que o código fique visualmente compreensível.
- **Comentação:** Comentar adequadamente, explicando o que cada rotina faz, o que retorna e o comportamento das variáveis. Comentar também como é feita a leitura das entradas.
- **Clareza:** Escolher um estilo de programação claro, evitando variáveis globais e modularizando o código.
- **Código DRY:** Manter o código DRY (Don't Repeat Yourself).

10.1 Código DRY

A sigla DRY é uma abreviação de Don't Repeat Yourself, que significa "Não Se Repita" em português, ou seja, é um princípio de programação de computadores que visa evitar a duplicação de código. Algumas formas de se aplicar esse princípio são:

- Criar funções ou métodos reutilizáveis e objetivos.
- Dividir o sistema em partes.
- Utilizar geradores de código, construtores automáticos e linguagens de script.

Códigos duplicados podem gerar problemas como:

- Necessidade de se gerenciar memória.
- Necessidade de se gerenciar ciclos de *runtime*.
- Aumento na probabilidade de *bugs*.
- Redução no desempenho da aplicação.
- Com a ocorrência de *bugs*, mais tempo gasto para se encontrar os erros e corrigí-los.