Introdução ao calango

Estrutura Básica

- O algoritmo possui uma estrutura básica composta da declaração do nome do algoritmo, e do bloco principal do algoritmo, conforme visto no código abaixo.

```
algoritmo nomeAlgoritmo;

    principal

        //código principal

fimPrincipal
```

Bloco Principal

- O bloco principal é delimitado pelas palavras ***principal*** e ***fimPrincipal***. Este bloco é o que será executado primeiro quando o algoritmo for interpretado. Os comandos que estiverem dentro deste bloco serão executados sequencialmente.
- Existem dois tipos de comentários: comentário de **linhas** e comentário de **várias linhas**. Tudo que estiver contido dentro de um comentário será prontamente descartado pelo interpretador.

Comentários

```
algoritmo nomeAlgoritmo;
/* aqui pode ser colocar
   várias linhas */
principal
        //comentário em uma única linha
fimPrincipal
```

Declaração de variáveis

- A declaração de variáveis, caso exista, é obrigatoriamente a primeira coisa a aparecer no algoritmo.
- Deve vir logo em seguida à abertura do bloco (seja ele o bloco principal, ou blocos de funções - como será visto adiante).

```
algoritmo nomeAlgoritmo;

    principal

        texto nome, sobrenome;

        inteiro idade;

        real peso;

        real altura;

fimPrincipal
```

Declaração de variáveis

- A tabela abaixo exibe os tipos de dados disponíveis para uso, junto com sua aproximação na linguagem C.

Calango	C
texto	char*
caracter	char
real	float

inteiro	int
lógico	0,1

- É possível criar vetores (arrays) no Calango. Na declaração de uma variável, basta adicionar colchetes com um número inteiro entre eles após o identificador da variável. Um exemplo no código à seguir.

```
algoritmo nomeAlgoritmo;
    principal
        texto nome, eventos[10];
        inteiro anos[10];
fimPrincipal
```

Atribuição de variáveis

- No vetor, para atribuir valores é necessário saber qual a posição no vetor quer atribuir. Como mostra o exemplo abaixo:

```
algoritmo atribuirVariaveis;
        principal
           inteiro anos[10];
           anos[0]=1;
          anos[1]=2;
fimPrincipal
```

Operadores e Comparadores

Operadores de texto	Descrição
+	concatenar textos

Operadores lógicos	Descrição
e	Operador lógico and
ou	operador lógico ou
nao	realiza a negação da operação

Comparadores	Descrição

==	comparação de igualdade
!=	comparação de diferença
>	comparação de maior
<	comparação de menor
<=	comparação de menor ou igual
>=	comparação de maior ou igual

Escrevendo e lendo dados

- Para que o algoritmo seja de fato útil ao usuário, recebendo e imprimindo dados para o usuário. Para que isso seja feito, faz-se uso de algumas funções embutidas.

- função ***escreva*** ou ***escreval***, ambas recebendo como parâmetro o que será impresso. Este valor pode ser um inteiro, texto, lógico, etc.
    - A diferença entre as duas funções é que a primeira apenas apresenta o valor e segue na mesma linha, enquanto que a segunda apresenta e pula para a próxima linha. O uso do operador “,” dentro da função ***escreva*** ou ***escreval***, realiza a concatenação dos textos. O código abaixo exemplifica algumas escritas.

```
algoritmo escrevendo;
        principal
           texto nome;
           inteiro idade;
          nome =”João”;
           idade=23;
           escreva(“Nome: ”); //apresenta e permanece na linha
          escreval(nome); // apresenta e salta linha
        escreva(“Idade: ”, idade);
        fimPrincipal
```

```
algoritmo lendo;
        principal
           texto nome;
           caracter sexo;
           inteiro idade;
        escreva(“Informe o Nome: ”);
           leia(nome);
```

```
        escreva("Informe a idade: ");
        leia(idade);
        escreva("Informe o sexo: ");
        leiaCaracter(sexo);
        escreval("nome: ", nome); // apresenta e salta linha
        escreval( "idade: ",idade);
      escreval( "sexo: ", sexo);
fimPrincipal
```

Tabela verdade

Tabela verdade é um dispositivo utilizado no estudo da lógica matemática. Com o uso desta tabela é possível definir o valor lógico de uma proposição, isto é, saber quando uma sentença é verdadeira ou falsa.

Conectivo	símbolo	valor lógico	valor
não	¬	negação	terá valor falso quando a proposição for verdadeira e vice-versa.
e	^	conjunção	será verdadeiro quando pelo

			menos uma das proposições for verdadeira.
ou	V	disjunção	será verdadeira quando pelo menos uma das proposições for verdadeira.

Exemplo negação ¬:

p	~p
V	F
F	V

Exemplo conjunção ^:

p	q	p^q
V	V	V
V	F	F
F	V	F
F	F	F

Exemplo disjunção V:

p	q	p v q
V	V	V
V	F	V
F	V	V
F	F	F

Exercício:

a) p ^ q ^ r
b) p V q V r
c) (p v q) ^ r
d) p v (q ^ r)
e) ¬p v (q ^ r)
f) p v (¬ (q ^ r))

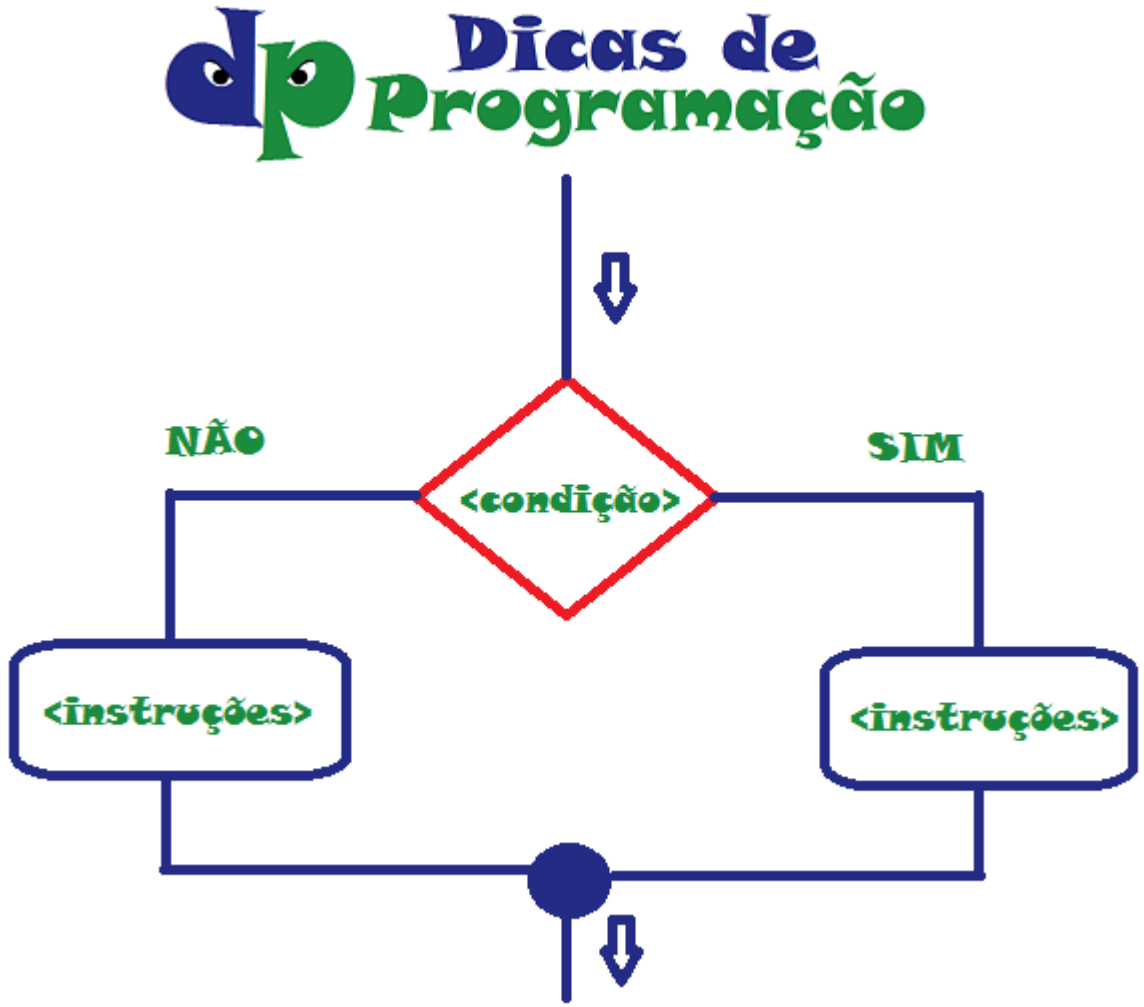
Condicionais e Repetições

Os condicionais e repetições (loops) são a maneira de dar vida ao algoritmo.

Se

A estrutura condicional mais simples é a estrutura ***se***. Ela segue a forma de ***se*** `(expressão)` ***entao .. fimSe***, onde a expressão deve resultar em um valor booleano (lógico).

SE *<expressão booleana>* ***ENTÃO***
<instruções a serem executadas caso a expressão booleana resulte em ***VERDADEIRO****>*
SENÃO
<instruções a serem executadas caso a expressão booleana resulte em ***FALSO****>*
FIM-SE



Um exemplo de uso básico desta estrutura é demonstrado no código abaixo.

```
#include <stdio.h>
int main(){
      int idade;
      idade=12;
       if (idade ==12){
            printf("igual a 12\n");
        }
       if (idade <=12) {
            printf ("menor o igual a 12\n");
       }
       if(idade < 12) {
            printf ("menor que 12\n");
       }
return 0;
}
```

if else

Em várias ocasiões é necessário executar blocos de forma alternativa. Nesses casos, podemos utilizar o comando `if-else` (que significa *se-senão* em português), cuja sintaxe é a seguinte:

```
int main(){
        inteiro idade;
        printf("Informe a idade:");
        scanf(“%d”, idade);
        if(idade >= 18){
                printf("maior de idade");
       }
        else if(idade>14) {
                printf("período de transição para adulto");
        }
        else if(idade > 12){
                printf("adolescente");
        }
        else if(idade > 10){
                printf("pré adolescente");
        }
        else { //(idade >= 0)
                printf("criança");
        }
fimPrincipal
```

Exercícios

1) Faça um algoritmo que simule um caixa eletrônico. O usuário informa o valor que deseja sacar, o caixa eletrônico verifica se o valor é menor que o saldo disponível e se, o valor é inferior ao saldo, informa que o saque será liberado, enquanto que, se o

valor for maior que o saldo, informar ao usuário que o saque excedeu o saldo na conta. Assuma um saldo de $10.000,00 de saldo disponível para o saque.

2) Faça um programa em C que leia os valores A, B, C e imprima na tela se a soma de A + B é menor que C.
3) Faça um programa que leia dois valores inteiros A e B se os valores forem iguais deverá se somar os dois, caso contrário multiplique A por B. Ao final de qualquer um dos cálculos deve-se atribuir o resultado para uma variável C e mostrar seu conteúdo na tela.
4) Encontrar o dobro de um número caso ele seja positivo e o seu triplo caso seja negativo, imprimindo o resultado.
5) Faça um algoritmo que, dados 3 números inteiros A, B e C, apresente qual dos 3 é o maior e qual é o menor.
6) Faça um programa que leia uma variável e some 5 caso seja par ou some 8 caso seja ímpar, imprimir o resultado desta operação.
7) O IMC – Índice de Massa Corporal é um critério da Organização Mundial de Saúde para dar uma indicação sobre a condição de peso de uma pessoa adulta. A fórmula é IMC = peso / (altura * altura). Elabore um programa que leia o peso e a altura de um adulto e mostre sua condição de acordo com a tabela abaixo: **IMC em adultos Condição:**
    a) Abaixo de 18,5 “Abaixo do peso”
    b) Entre 18,5 e 25 “Peso normal”
    c) Entre 25 e 30 “Acima do peso”
    d) Acima de 30 “obeso”

Estrutura condicional If e else na linguagem C

```c
#include <stdio.h>

int main(){
int a;
printf("Digite um numero:");
scanf("%d", &a);
if(a > 18)
{
	printf("Sou maior de idade\n");
}
else
{
	printf("Sou menor de idade\n");
}

return 0;
}
```

Estruturas de repetição

Após a leitura deste capítulo, você será capaz de:

- Identificar a estrutura repetitiva de um problema.
- Conhecer os comandos e escrever programas em C que realizam repetições.
- Aprender a testar e simular programas com comandos de repetição.

Na linguagem C existem três comandos de repetição: **while**, **do-while** e **for**.

Introdução

Sem dúvida alguma, a capacidade de repetir operações foi decisiva para consolidar os computadores como ferramentas de resolução de problemas. Uma vantagem clara é que na repetição realizada por um computador todas as operações são executadas do mesmo jeito. Nesta unidade, estudaremos um comando que pode ser usado para executar uma ou mais instruções repetidamente, até um número desejado de vezes. As repetições são organizadas em ciclos. No início de cada ciclo, uma expressão lógica é testada para determinar se a repetição deve prosseguir ou não.

Para entender como isso acontece, vamos considerar um exemplo. Suponha que temos que escrever um programa que mostra os números 1, 2, 3 e 4 na tela do computador. Usando o que aprendemos até o momento, chegamos à solução a seguir.

```c
#include <stdio.h>
int main (){
   int i=1;
   printf("%d\n", i);
   printf("%d\n",(i+1));
   printf("%d\n",(i+2));
   printf("%d\n",(i+3));
return 0;
}
```

Imagine agora que temos que imprimir todos os números entre 1 e 1000
Seria algo fácil de se fazer, todavia, levaria um bom tempo até concluir a atividade.

Afinal, como podemos resolver isso de forma simplificada?

Para isso, vamos precisar usar laços de repetição.

Comando while

O comando while é utilizado para executar um bloco de comandos enquanto uma condição é satisfeita. Importante: a repetição deixa de ser executada quando a condição é falsa. A estrutura do comando é apresentada a seguir.

```c
   while (condição)
   {
       // comandos a serem repetidos
   }
   // comandos após o 'while'
```

O funcionamento é o seguinte:

1. Testa a condição;
2. Se a **condição** for **falsa** então pula todos os comandos do bloco subordinado ao **while** e passa a executar os comandos após o bloco do **while**.
3. Se **condição** for **verdadeira** então executa cada um dos comandos do bloco subordinado ao **while**.
4. Após executar o último comando do bloco do **while** volta ao passo 1.

O comando **while** deve ser usado sempre que:

- não soubermos exatamente quantas vezes o laço deve ser repetido;
- o teste deve ser feito antes de iniciar a execução de um bloco de comandos;
- houver casos em que o laço não deva ser repetido nenhuma vez.

Três perguntas principais nos ajudam a utilizar este comando:

- O que deve ser repetido?
- Quantas vezes devem se repetir?
- Qual condição pode ser utilizada para representar essa repetição?

Agora que já conhecemos o comando while, podemos voltar ao problema da impressão de todos os números entre 1 e 1000:

```c
#include <stdio.h>
int main (){
   int i=0;
   while (i < 1000)
   {
       i++;// equivalente a i=i+1
       printf("%d\n",i);
   }
 return 0;
}
```

IMPORTANTE: Identificar o que se repete e o que muda é a tarefa chave nesse processo.

Desafio: Encontrando o maior número digitado

O objetivo agora é escrever um programa que lê um número inteiro n e, em seguida, lê n valores inteiros e mostra o maior deles na tela.

Após a leitura de n igual a 10, por exemplo, seu programa deve ler 10 (dez) números inteiros. Estes dez números poderiam ser, digamos, 5, 4, 1, 7, 45, 12, 54, 65, 23, 31. Seu programa deveria imprimir, neste caso, o número 65 (o maior número digitado pelo usuário).

Para resolver este problema, vamos dividir o problema em partes bem pequenas, resolver cada uma delas separadamente e, depois, juntar tudo.

1- O **primeiro** subproblema desta atividade diz respeito à leitura do valor de n.

2- O Segundo subproblema desta atividade diz respeito a iterar pelos n números digitados pelo usuário.

3- O terceiro subproblema desta atividade diz respeito a regra de negócio para pegar o maior valor digitado pelo usuário.

Desafio: Laço de repetição controlado

```c
#include <stdio.h>
int main ()
{
   int contador;
   char continua;
   continua = 's';
   contador = 0;
   while (continua == 's')  // enquanto for igual a 's'
   {
       // comandos a serem repetidos
       printf("Repentindo....\n");
       contador = contador + 1;
       printf("Tecle 's' se deseja continuar?");
       scanf(" %c",&continua);
   }// fim while
   if (contador == 0)
   {
       printf("O bloco NAO foi repetido.");
   }
   else
   {
       printf("O bloco foi repetido %d vezes", contador);
   }
return 0;
}
```

Desafio: Média de idade de um grupo de usuários

```c
#include <stdio.h>
int main ()
{
   int soma, quantidade, idade;
   float media;
   soma = 0;
   quantidade = 0;
   idade = 0;
   while (idade != -1)
   {
       // comandos a serem repetidos
     printf("Digite a idade da pessoa %d ou (tecle -1 se quiser encerrar):", quantidade+1);
       scanf("%d", &idade);
       if (idade >=0)
       {
           soma = soma + idade;
           quantidade= quantidade + 1;
       }
   }//fim while
   // Calcular a media do grupo
   if (quantidade >0)
   {
   media = (float) soma / quantidade;
   printf("A media de idade das %d pessoas eh: %.2f\n", quantidade, media);
   }
   else
   {
       printf("Nenhum dado foi informado.\n");
   }
  return 0;
}
```

Comando do-while

O comando **do-while** permite que um trecho de programa seja executado **enquanto** a condição for **verdadeira**. A forma do comando **do-while** é dada abaixo:

```
do
{
     // comandos a serem repetidos
} while (condição);
```

O funcionamento:

- Processa os comando dentro do bloco **do-while**;
- Testa a condição;
- Se a **condição** for **falsa** então executa o comando que está logo após ao **do-while**
- Se a **condição** for **verdadeira** então volta ao passo 1.

O comando **do-while** deve ser usado sempre que:

- quando não soubermos exatamente quantas vezes o laço deve ser repetido;
- o teste deva ser feito **depois da execução** de um bloco de comandos;
- o bloco de comandos deve se **executado pelo menos 1 vez;**

```
#include<stdio.h>
int main()
{
 char nome[30];
 float nota1=0,nota2=0,media=0;
 int resp;
  do
 {
   printf("Digite o nome do aluno:");
   scanf(" %s",nome);
   printf("Digite a primeira nota: ");
   scanf("%f",&nota1);
   printf("Digite a segunda nota: ");
   scanf("%f",&nota2);
   media = (nota1 + nota2)/2;
   printf("A Media do aluno %s foi = %f\n",nome, media);
   printf("Digite 1 para continuar ou 2 para sair:");
   scanf("%d", &resp);
  }while (resp==1);
```

```
 return 0;
}
```

Comando for

Permite que um trecho de programa seja executado **um determinado número de vezes**. A forma do comando **for** é dado como segue:

```
for (comandos de inicialização;condição de teste;incremento/decremento)
{
   // comandos a serem repetidos
}
```

O funcionamento:

1. Executa o comando de **inicialização**;
2. Testa a condição;
3. Se a **condição** for **falsa** então executa o comando que está logo após o bloco do **for**.
4. Se **condição** for **verdadeira** então, execute os comandos que estão no bloco do **for**;
5. Executa os comandos de **incremento/decremento**;
6. Volta ao passo 2.

O comando **for** deve ser usado sempre que:

- **soubermos exatamente quantas vezes o laço deve ser repetido**;
- o teste deve ser feito **antes da execução** de um bloco de comandos;
- houver casos em que o laço **não deva ser repetido nenhuma vez**.

Observações:

1. Os comandos de inicialização é executado **apenas 1 vez**;
2. O contador é incrementado/decrementado **imediatamente após** execução do bloco;
3. O teste é feito sempre antes do início da execução do bloco de comandos.

```
#include<stdio.h>
int main()
{
   int contador;
   for (contador = 0; contador < 10; contador = contador+1)
   {
         printf("Valor do Contador eh: %d\n",contador);
```

```
    }
    printf("Valor do Contador após o laço eh: %d\n",contador);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    int contador;
    for (contador = 10; contador > 0; contador = contador-1)
    {
        printf("Valor do Contador eh: %d\n",contador);
    }
    printf("Valor do Contador apos o laco eh: %d\n",contador);
}
```

Desafio: Escreva um programa em C, que compute a média de idade de um grupo de 10 usuários usando laço de repetição **for**.