Resumão C2O2 - P2

Definição de Código e Programa.

-> Um programa é um meio de uma pessoa realizar ações (aplicações) para resolver problemas, pesquisas, cadastro ou até mesmo uma rede social.
-> O código está contido no programa de forma que o código dita as ações que o programa irá fazer.
  -> Existem várias linguagens de códigos, essas linguagens possuem seu próprio modo de sintaxe, organização e funções.
  -> Este resumo abordará a linguagem C/C++.
-> É importante ressaltar que o código precisa ter uma lógica entre suas funções e comandos para que o programa possa ser executado corretamente e sacie as exigências/atribuições que o programa se propôs a fazer.

Variáveis

-> Variáveis são responsáveis por guardar dados que são introduzidos durante o programa.
-> As variáveis podem ser vários tipos:

  int => números inteiros
  double => números de ponto flutuante (decimais), mas com maior precisão
  float => número de ponto flutuante com menor precisão
  char => caracteres
  bool => true ou false (1 ou 0)

-> Variáveis tem que começar com letras; não podem conter espaço, caso queira separar use o underline ("_"); letras maiúsculas se diferenciam de letras minúsculas; não pode usar como variável uma sigla que representa uma função:

  1bc -> não pode porque começa com nº.
  do -> não pode porque é uma função
  soma ≠ SOMA ≠ SoMA ≠ somA ≠ soMa ≠ sOma ≠ SOma ≠ SOmA
  gera menor -> não pode porque contém espaço; deveria usar o underline para separar as palavras.
  b1c, DO, gera_menor -> exemplo de variáveis permitidas.

-> Toda variável tem que ser declarada.
-> Caso a declaração de variável esteja incorreta a variável não armazenará o dado corretamente.
-> Lembrando que há mais tipos de variáveis, apresentei os mais usados.

Biblioteca

→ Biblioteca é uma coleção de subprogramas prontos que permitem que o programa leia no código uma determinada função e o execute.

→ Para importar uma biblioteca em C++ devemos colocar o comando:

```
#include <nome_biblioteca>
```

→ Algumas bibliotecas padrão:

#include <iostream>

→ abilita a entrada e saída de dados.

→ Portanto permite o programa mostrar e receber dados.

→ Isso ocorre por meio das funções:

`cin >> nome_variável;` → permite a entrada de um dado à variável posta após o >>.

`cout << "texto";` → permite a saída de uma mesagem que esteja dentro das aspas, caso queira mostrar o dado de uma variável é só colocar a variável após o <<.

#include <cmath> ou #include <math.h>

→ permite a realização de operações matemáticas mais complexas

→ lembrando que o "+", "-", "÷" e "x" não precisam da declaração dessa variável para serem usados no código e que: multiplicação é representada por asterisco ("*") e divisão por barra ("/").

→ Aqui estão algumas das operações:

- floor = arredonda o valor para baixo
- ceil = arredonda o valor para cima.
- sqrt = raiz quadrada
- cbrt = raiz cúbica
- pow = potenciação

#include <iomanip>

→ permite que seja definido o nº de casas que serão apresentadas após a virgula.

→ Seu comando é:

```
cout << fixed << setprecision(2) << soma;
```

↳ neste caso mostrará o valor da variável soma com duas casas decimais.

→ Todos os dados que tiverem que aparecer no monitor e que tiveram esse comando precedindo eles terão as casas decimais mostradas de acordo com o que ele definiu.

Comentários

-> Comentários servem para que outras pessoas, que não ajudaram ou fizeram o programa, saibam o que está acontecendo no código.

// comentário
/* comentário */ } são as duas formas de representar o código.

↳ este pode ser escrito em mais de uma linha enquanto o // só pode ser escrito em uma.

-> Os comentários não atrapalham na execução do programa porque o programa não os lê, mas o código os guarda.

```
cin >> x;   // o programa pedirá a entrada do valor de x.
```

↳ exemplo de um comentário que serve para explicar o que o comando cin está fazendo.

Informações importantes

-> Para que um programa possa mostrar caracteres únicos da língua portuguesa, como acentos, é preciso usar a função abaixo para abilitar o idioma português:

```
setlocale (LC_ALL, "Portuguese");
```

-> Relacionado a operações matemáticas tem-se:

++ → incrementa um ao valor da variável
-- → decrementa um ao valor da variável
== → Comparar igualdade
!= → Comparar desigualdade
! → tudo o que vier após esse símbolo é tomado como não.
% → resto de divisão.

-> Para pular uma linha na saída de dados pode-se usar:

```
endl  ou  \n
```

-> Para fazer a leitura de um texto é necessário utilizar o comando:

```
cin.getline(nome, tamanho);
```

-> Se antes de usar o comando cin.getline alguma entrada de dados foi feita, é necessário utilizar o comando:

```
cin.ignore();
```

-> Sempre, após colocar as bibliotecas, colocar o comando abaixo para não ter que colocar std antes de cada cin|cout:

```
using namespace std;
```

Visão geral do código

```
//no início são adicionada todas as bibliotecas
#include <iostream>

using namespace std;
/* função principal. É onde ficam as instruções do programa*/
int main ()
{ //início

    //declaração de variáveis


    //instruções do programa

    retun 0; /*valor que a função retorna ao compotador*/
} //fim
```

* Identação é o nome dado à organização que um programa tem que apresentar para que não ocorre poluição visual do código e que seja simples de analisá-lo.

Estruturas de decisão

-> basicamente delimitam se uma condição for verdadeira ocorrerá um comando e caso não seja ocorrerá outro comando ou o programa acaba seguindo a execução

- **If simples**

-> O comando if será sempre usado para analisar uma situação e tomar uma decisão.

-> O if simples faz o seguinte:

- Se tal condição for verdadeira —> faça isso
- Se não —> não faça nada.

-> Sintaxe:

```
if (condição)
    comando;
```

—> caso tenha que ser feito mais de um comando:

```
if (condição) {
    comando 1;
    comando 2;
    comando n;
}
```

• If composto

→ O if composto apresenta 2 ações possíveis:

- se tal condição for verdadeira → comando 1
- se não → comando 2.

→ Sintaxe:

```
if (condição) {
        comando 1;
}
else {
    comando 2;
}
```

} observe que o else não acompanha condição porque somente o if pode desencadear uma condição.

• If-else aninhado

→ Nada mais é do que um comando if simples ou composto dentro de um if ou else; Sua sintaxe é:

```
if (condição 1) {
    if (condição 2)
        comando 1;
}
else {
    comando 2;
    if (condição 3)
        comando 3;
    else
        comando 4;
}
```

* Lembrando que quando uma condição já é satisfeita não faz as outras operações; as outras condições só são analisadas caso a anterior não tenha sido satisfeita.

• If-else encadeado

→ Nada mais é do que um comando if seguido de um ou vários comandos else if.

→ Sintaxe:

```
if (condição 1) {
    comando 1;
}
else if (condição 2) {
    comando 2;
}
else if (condição 3) {
    comando 3;
}
```

* Comando else if pode ser acompanhado de condição porque tem o if.

• Operadores Comparativos

(==) Igualdade
(>) maior
(<) menor
(>=) maior igual
(<=) menor igual
(!=) diferente
(&&) E lógico
(||) OU lógico
(!) Não lógico.

→ São usados dentro das condições

• Switch

→ Comando switch irá verificar se uma variável do tipo int ou char é igual a uma constante.

→ Sua condição é somente de igualdade

→ Sintaxe:

```
switch (variável) {

    case 1: //testa se variável == 1
        //comandos
        break; /* o break é usado para sair do case assim que
                  executar os comandos da condição */
    case 2: //testa se variável == 2
        //comandos
        break;
    default:
        /* o default será executado caso nenhuma das condições dos
        cases for satisfeita; o default é opicional no switch */
        //comandos
}
```

→ Caso a analise de igualdade for relacionada à caracter, entre com a condição entre aspas simples (' ').

→ O switch consegue fazer comparação apenas com caracteres unitarios.

→ Lembrar-se sempre de colocar dois pontos (:) após a condição do case.

Estruturas de Repetição

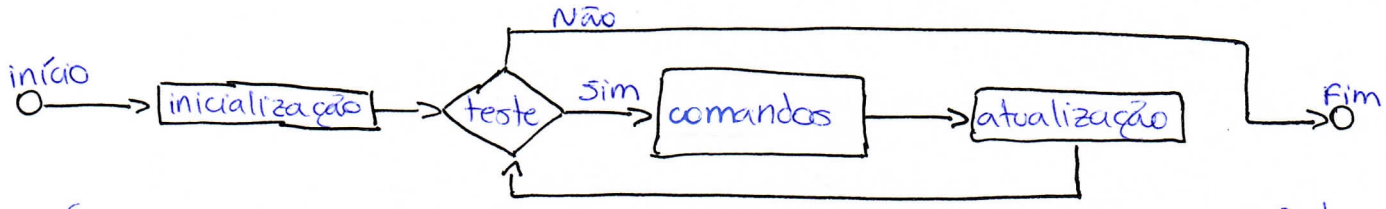
→ não só permitem executar o bloco de comandos sob determinadas condições, mas como também de repetir o mesmo bloco quantas vezes for necessário, até que uma certa condição seja satisfeita.

• For

-> normalmente utilizada quando sabemos previamente a quantidade de vezes que os comandos dentro dela devem ser executados.

-> Sua declaração crítica é baseada em uma variável que será incrementada ou decrementada até chegar a um limite definido.

-> Funcionamento:



* É na atualização que ocorre o incremento/decremento da variável.

-> Sintaxe:

```
for (inicialização; teste; incremento) {
    comandos
}
```

• While

-> O comando while é utilizado quando não sabemos o número de vezes que os comandos devem ser repetidos.

-> Um for pode ser modificado para virar um while, e vice-versa

-> Sintaxe:

```
While (condição_de_parada) {
    comandos
}
```

ocorrem por erros na construção da condição

-> Quando não ocorre nenhum comando é porque a condição de parada não foi satisfeita desde o início.

-> Quando ocorre a execução infinita é porque a condição sempre é satisfeita.

• Do-while

-> O comando do-while segue o mesmo princípio do comando while, sendo que esse comando executa pelo menos uma vez os comandos e depois analisa a condição para haver ou não repetição dos comandos.

-> Sintaxe:

```
do {
   comandos
} while (condição_de_parada);
```

-> Existe uma definição chamada flag, a qual serve como controle de execução do comando do-while.

-> Portanto se a flag for satisfeita não ocorrerá mais o comando do-while.