RESUMO DE OPERADORES E ESTRUTURA CONDICIONAIS

Anteriormente, quem leu o resumo anterior sobre fluxograma e variáveis viu um dos tipos de operadores, o aritmético onde mostro que + (soma), - (subtração), *(multiplicação), / (divisão) e um novo que não foi comentado no outro resumo que é a %, esse significa módulo (resto).

Ele é usado geralmente para ficar em um intervalo que a pessoa escolhe, já que é o resultado do resto da divisão de um número pelo outro. Exemplo:

```
Resto = 8 % 3;
```

Isso resulta no resto de 2. Portanto esse operador, permite (quando utilizado), ficar dentro de um intervalo que deseja. Por exemplo, se quer que os resultados do número inteiro fiquem no intervalo entre 0 a 9, basta declarar o valor de x como:

```
x = x % 10;
```

INCREMENTO E DECREMENTO UNITÁRIO:

O incremento é importante nos problemas que envolvam o uso de contagem de laços.

Incremento:

```
i = i + 1;
```

Existem outras formas de escrever os incrementos no código, eles são:

```
i++;
```

```
++i;
```

O i++ é o pós-fixado ou pós-incremento, ele incrementa depois da leitura da variável, nesses exemplo o i é uma variável de valor inteiro. Já o ++i ele incrementa antes da leitura da variável.

Exemplo:

```
i = 2 + 1;

b= i ++ * 2;
```

O resultado vai dar 6, já que o sinal usado no i é para pré-incremento, ou seja, vai fazer a conta com o número de i = 3 multiplicado por 2, depois de ter executando a segunda operação o i será incrementado, valendo 4.

```
i = 2 + 1;

b = ++i * 2;
```

O resultado desse vai ser 8, pois o resultado do i = 3 (quando executa a soma) e como o i está como pós-incremento ele se torna 4 que multiplica por 2 e dá o resultado de 8.

Como tem o incremento, existe o decremento unitário, tem a mesma lógica, porém em vez de acrescentar ele diminui. Pré-decremento é --i e o pós-incremento é i--. Essas são as formas mais usadas nos códigos, quando usadas, mas por extenso significa i = i – 1;

Como na programação existe outra forma de usar o incremento e o decremento, para as contas básicas de soma, subtração, multiplicação, divisão e resto também existe, e são chamados de atribuição.

Demonstração:

- Símbolo da soma: += a sintaxe fica assim: a+=b e isso substitui a=a+b;
- Símbolo da subtração: -= a sintaxe fica assim: a-=b e isso substitui a=a-b;
- Símbolo da multiplicação: *= a sintaxe fica assim: a*=b e isso substitui a=a*b;
- Símbolo da divisão: /= a sintaxe fica assim: a/=b e isso substitui a=a/b;
- Símbolo do módulo (resto): %= a sintaxe fica assim: a%=b e isso substitui a=a%b;

Isso é possível pois a variável que você quer que receba o valor é a que quer somar, subtrair ou qualquer uma das operações acima, e se usasse a mesma variável suas vezes, poderia ficar mais difícil de entender o código futuramente ou até mesmo se perder na hora de programar, fora que ficaria muito poluído, sendo melhor assim, usar essa forma de atribuição que todos os programadores entendem e deixa o código mais resumido e limpo.

Importante lembra que como na matemática, na hora de resolver as contas, tem suas prioridades, para assim chegar no resultado correto. Na programação também existe, então quando não for usar parênteses para especificar qual conta deseja ser resolvida primeiro, se atende pela ordem de prioridade que irei mostrar logo abaixo, pois se não fizer isso a conta pode estar certa, mas retornando o valor errado.

Irá estar na ordem do que é resolvido primeiro, com seus respectivos símbolos:

- Chamada de função [f()];
- Pré-incremento [++];
- Pré-decremento [--];
- Multiplicação [*];
- Divisão [/];
- Módulo (resto) [%];
- Adição [+];
- Subtração [-].

OPERADORES RELACIONAIS E LÓGICOS:

Quando tem operador que resulta em somente dois valores [Verdadeiro / Falso], na linguagem C seria 0 para falso e 1 para verdadeiro, já que nessa linguagem não existe verdadeiro e falso.

Os operadores relacionais são:

- menor que [<];
- menor ou igual que [<=];
- maior que [>];
- maior ou igual que [>=];

- diferente de [!=];
- igual a [= =].

Os operadores lógicos são:

- Conjunção E [&&]: usado para quando a condição tem que ser essa **e** a outra para ser verdadeira, por exemplo, P&&Q.
- Disjunção OU [||]: já está basta uma das condições estarem corretas segundo a condição proposta para que execute o desejado, uma demonstração de como usar P||Q.
- Negação NÃO [!]: esse é usado para negar alguma condição, nos códigos geralmente é usado como flag de um laço. Uma das maneiras de como utilizar esse operador é !Q.

Esses operadores relacionais e lógicos são geralmente usados para expressar alguma condição para executar ou não uma estrutura condicional que iremos abordar abaixo.

ESTRUTURAS CONDICIONAIS:

Pode ser chamado de estrutura de decisão, essas decisões já vão ser pré-estabelecida pela pessoa que montar determinado código. Com isso já deixando o fluxo do programa determinado segundo as informações que uma pessoa informar.

Elas são o **if-else** ou **switch case**, vai depender qual irá melhor se encaixar com o problema proposto, porém tem vezes que será usado as duas em conjunto, em problemas mais complexos.

Vamos começar explicando a estrutura do if else:

```
if (<operação lógica>)
        <comando>;
else
        <comando>;
```

Essa é uma estrutura básica do if else, vale lembrar que pode conter mais de um comando, ou seja, pode escrever seu código no lugar onde está escrito comando até atender ao que está pedindo o problema, porém quando tiver mais de um comando é essencial que contenha {} para mostrar onde começa e onde termina os comandos do if e do else.

Por exemplo:

```
if (<operação lógica>) {
        <comando>;
        .
        .
        <comando>;
}
else {
        <comando>;
        .
        .
        <comando>;
}
```

No entanto, quando tem mais de dois caminhos, tem que acrescentar mais uma coisa na estrutura que é um else if(<operação lógica>), a estrutura é a mesma mas fica entre o if e o else. E pode acrescentar a quantidade que precisar.

O if else pode ser lido como “se a operação do if for verdadeira executa, caso contrário passa para o próximo else if ou else”, ou seja, o if só vai ser executado quando a operação logica (que é escrito através dos operadores relacionais ou lógicos) forem verdadeiros. Nessa linha de raciocínio, é percebido que se um deles for executado, o restante é ignorado e o código continua. Então tem que pensar bem na estrutura para que tudo que for pedido seja executado.

Importante lembrar que um if pode estar dentro de outro if, um exemplo em um código básico:

```
#include <stdio.h>

int main() {
   int numero1 = 5;
   int numero2 = 10;

   if (numero1 > 0) {
      printf("O numero1 e positivo\n");

      if (numero2 > 0) {
         printf("E o numero2 tambem e positivo\n");
      } else {
         printf("Mas o numero2 e negativo\n");
      }
   } else {
      printf("O numero1 e negativo\n");
   }

   return 0;
}
```

Já o switch case é usado geralmente quando uma variável pode assumir valores exatos e disjuntos entre si. Geralmente é usado quando a pessoa tem que escolher alguma opção a ser executada. Desta forma, dependendo a opção escolhida é o comando executado.

A estrutura dessa função:

```
switch (<variável>) {
case <valor>:
        <comando>;
```

```
            ...
            <comando>;
            break;

        case <valor>:
            <comando>;
            ...
            <comando>;
            break;

        case <valor>:
            <comando>;
            ...
            <comando>;
            break;

        case <valor>:
            <comando>;
            ...
            <comando>;
            break;
        default:
            <comando>;
            ...
            <comando>;
}
```

Um exemplo em um código, para ficar mais claro como funciona:

```
#include <stdio.h>

int main() {
    int escolha;
    printf(“Escolha um número de 1 a 4:”);
    scanf(“%i”, &escolha);
    switch (escolha) {
        case 1:
            printf("Escolheu a opcao 1\n");
            break;
        case 2:
            printf("Escolheu a opcao 2\n");
            break;
        case 3:
            printf("Escolheu a opcao 3\n");
            break;
        case 4:
            printf("Escolheu a opcao 4\n");
            break;
        default:
            printf("Opcao invalida\n");
            break;
```

```
    }


    return 0;
}
```

No caso do exemplo, de a pessoa escolhesse um número que fosse de 1 a quatro iria entrar no seu respectivo case, caso escolhesse um número maior ou menor que pediu iria entrar no default. Esse exemplo foi básico, só para entender melhor como funcionava a estrutura.