Aula 09 - Operadores de auto incremento e auto decremento, atribuição múltipla, atribuição na declaração, atribuição resumida e algumas regras de boa programação.

1) Operadores de auto incremento ++ e auto decremento --

A operação de somar ou subtrair 1 de um contador ou de uma variável qualquer dentro de um programa é muito comum.
Os operadores ++ e – fazem isso diretamente sem a necessidade do comando de atribuição.

`i = i + 1;` é equivalente a `i++;`
`i = i – 1;` é equivalente a `i--;`

Podem ser usados à esquerda ou à direita.

À esquerda – incrementa/decrementa e a seguir usa o valor
À direita – usa o valor e a seguir incrementa/decrementa

`i++;` ou `++i;`
`i—-;` ou `--i;`

Outros exemplos:

`b = ++a;` equivalente a `{a = a + 1; b = a;}`
`b = a++;` equivalente a `{b = a; a = a + 1;}`
`b = --a;` equivalente a `{a = a – 1; b = a;}`
`b = a--;` equivalente a `{b = a; a = a - 1;}`

Esses operadores quando usados no meio de comandos, confundem um pouco.

É conveniente usá-los em sua forma mais simples.

Exemplos confusos:

1) `a = 5; b = a * (a++);`
`b` fica com 5*5=25 e `a` fica com 6

2) `a = 5; b = a * ++a;`
`b` fica com 5*6=30 e `a` fica com 6

3) `a = 5; b = a * a++;`
`b` fica com 5*5=25 e `a` fica com 6

Veja o exemplo abaixo:

P39) trechos de programa que imprime os números de 1 a n

```
#include <stdio.h>
#include <stdlib.h>
/* dado n>=0 imprime os números de 1 a n de várias formas */
int main() {
 int n,    /* número dado */
     i;    /* contador */
 /* ler o n */
 printf("digite o valor de n:");
 scanf("%d", &n);

 /* exemplo 1 */
 printf("\n");
 i = 1;
 while (i <= n) {printf("%5d", i); i++;}

 /* exemplo 2 */
 printf("\n");
 i = 1;
 while (i <= n) printf("%5d", i++);

 /* exemplo 3 */
 printf("\n");
 i = 0;
 while (i++ < n) printf("%5d", i);

 system("PAUSE"); return 0;
}
```

2) Atribuição múltipla

A atribuição (=) também é um operador. Não é operador aritmético, porém é um operador de movimento de dados. Como tal, tem uma prioridade associada, a qual é menor que os operadores aritméticos. Assim, num comando de atribuição, primeiro é calculada a expressão aritmética do lado direito, para depois efetuar a atribuição. Exemplos:

```
a = b;
a = b + c;  /* primeiro efetua a soma depois a atribuição */
a = b + c + d + e; /* idem */
```

Assim, como a atribuição também é operador, podemos ter um comando do tipo:

```
a = b = 0;  /* zera a e b */

a = b = c = d = e = 1; /* atribui 1 a a, b, c, d e e */

a = b = (c+d); /* calcula c+d e atribui a b e depois a a */
```

Devido à prioridade, as atribuições são realizadas da direita para a esquerda.

3) Declaração com atribuição inicial de valor

Ao fazermos uma declaração é possível atribuirmos um valor inicial a esta variável declarada.

```
int main() {
  int a = 0;
  int b = 31;
  int x = 1, y = 2, z = 3;
```

O mesmo ocorre com os outros tipos de variáveis bool, char, short, long, float, double, arrays e strings. Quando virmos cada um dos tipos, veremos a forma de atribuir valor inicial.

Atribuição resumida

Outra construção bastante comum em programação é a mesma variável ocorrer dos dois lados de um comando de atribuição.

Atribuição	Forma resumida
`A = A+5;`	`A+=5;`
`A = A-5;`	`A-=5;`
`A=A+B;`	`A+=B;`
`A=A-B;`	`A-=B;`
`A=A*B;`	`A*=B;`
`A=A/B;`	`A/=B;`
`A=A%B;`	`A%=B;`

Resolvendo novamente o P13:

P13) Dado N > 0 e uma seqüência de N números calcular a soma dos elementos da seqüência.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int N, soma=0, x, cont=1;
   printf("entre com o valor de N:");
   scanf("%d", &N);

   while (cont <= N) {
           printf("entre com mais um valor:");
           scanf("%d", &x);
           soma += x;
           cont++;
   }
   printf("\nvalor da soma:%10d", soma);
   system("PAUSE");
   return 0;
}
```

4) Algumas regras gerais de boa programação

a) Comentários

Sempre coloque comentários em seus programas. Não existem regras definidas de quantidade e qualidade de comentários. É tudo bom senso. Os comentários devem explicar o que faz cada trecho do programa de forma clara. Em especial, nas declarações de variáveis ou funções, descreva resumidamente qual o objetivo de cada elemento declarado.

b) Identação

Identar um programa é escrevê-lo de uma forma gráfica, de maneira que a observação do texto do programa dê alguma idéia de sua estrutura. A identação também é subjetiva, mais algumas regras parecem ser de senso comum.

- Abre chaves { e fecha chaves } alinhados sempre de uma mesma forma nos comando. Alguns programadores preferem { e } na mesma coluna e outros preferem alinhá-los com o comando em uso.
- O else embaixo do if correspondente, embora alguns o preferiam levemente deslocado em relação ao if.
- Os comandos de um mesmo bloco alinhados na mesma vertical.
- Blocos com hierarquia menor, internos aos de hierarquia maiores.
- Deixar espaços em branco no meio de expressões e comando de modo a torná-los mais legíveis. Exemplos:
  `while(N<=0||M<=0) ...` é pior que **`while (N <= 0 || M <= 0)`**
  `a=b%c+1 ...` é pior que **`a = b % c + 1`**

c) Nomes mnemônicos de variáveis

O nome da variável tem que dar uma idéia da função desta variável. Não faz sentido, uma variável importante dentro de um programa ter um nome simples tal como **`i`**. Já um contador de for usado várias vezes no programa pode se chamar **`i`**.

d) Programação com clareza

Use sempre que possível comando simples em seu programa. Isso é especialmente útil em C onde os programas quando escritos usando certas abreviaturas de comandos combinados entre si produzem programas muitas vezes ilegíveis, embora corretos.
Quando for necessário usar algum truque de programação, procure esclarecer o truque nos comentários.

e) Consistência dos dados

Faça sempre consistência dos dados lidos, procurando repetir a entrada, se o dado for inconsistente, após imprimir a mensagem de erro.
Exemplos:

```
/* ler um valor inteiro x entre 0 e 999 */
printf("\nentre com x (0 <= x <= 999):");
scanf("%d", &x);
while (x < 0 && x > 999) {
```

```
   printf("valor fora dos limites");
   /* ler novamente */
   printf("\nentre com x (0 <= x <= 999):");
   scanf("%d", &x);
}
/* o valor de x lido está nos limites pedidos */
```

Outra forma:

```
/* ler um valor inteiro x entre 0 e 999 */
x=-1; /* truque só para entrar a primeira vez na repetição */
while (x<0 && x>999) {
  printf("\nEntre com valor entre 0 e 999");
  scanf("%d", &x);
}
/* Neste ponto x está dentro dos limites pedidos */
```

Outras formas com construções que serão vistas à frente.

Outra forma:

```
/* ler um valor inteiro x entre 0 e 999 */
while (1)
  {printf("\nentre com x (0 <= x <= 999):");
   scanf("%d", &x);
   if (x >= 0 && x <= 999) break;
   printf("valor fora dos limites");
  }
/* o valor de x lido está nos limites pedidos */
```

Outra forma:

```
/* ler um valor inteiro x entre 0 e 999 */
for ( ; ; )
  {printf("\nentre com x (0 <= x <= 999):");
   scanf("%d", &x);
   if (x >= 0 && x <= 999) break;
   printf("valor fora dos limites");
  }
/* o valor de x lido está nos limites pedidos */
```

f) Pedido de entrada de dados – clareza na entrada de dados

Antes de ler algum dado, sempre imprimir um esclarecimento sobre o dado que está sendo solicitado. Se o seu programa lê somente um ou dois dados não tem problema, mas imagine um programa que precisa ler dezenas ou centenas de dados.

g) Clareza na saída de dados

A saída do programa é muito importante. Bons programas com saída muito simples e pouco clara causam má impressão. Procure sempre imprimir frases que indiquem claramente o que está sendo impresso. Por outro lado programas com excesso de saídas e frases repetitivas também causam má impressão. Procure sempre obter um equilíbrio, pensando que quem irá ler a saída não é apenas você que fez o programa, mas também outros usuários.

h) Clareza no algoritmo

Para isso não há regras. Existem bons algoritmos pouco claros e algoritmos claros e ruins. Por ruim aqui se entenda que gastem mais memória que o necessário ou que consumam muito tempo. Muitas vezes, entretanto com um pouco mais de ineficiência se produz algoritmos muito mais claros. Novamente procure pensar em que quem lerá o programa não é só você.