5 - COMANDOS DE CONTROLE DE PROGRAMA

Em C existem os comandos de decisões, os comandos de iteração (ou de laços) e os comandos de desvios.

5.1 - Comandos de decisões

Executam testes para tomar decisões. São eles: *if*, o *if-else* e o *switch*.

5.1.1 - Comando IF

- Comando de decisão simples.
- Se o resultado da expressão for verdadeiro, os comandos serão executados, caso contrário, nada é feito.

Sintaxe:

```
if (<expressão>)
        <comando(s)>
```

Exemplo:

```
/* testeif1.c */
#include <stdio.h>
main( )
{
   char ch;
   ch = getchar( );
   if  (ch == 'p')
      printf ("Você pressionou a tecla p \n");
}
```

- Podemos ter instruções múltiplas no corpo do comando if. Se existir mais de uma instrução no corpo do if, deve-se colocar chaves:

```
{
  - - -
  - - -
}
```

Exemplo:

```
/* testeif2.c */
#include <stdio.h>
main ( )
{
   if ( getchar( ) == ‘p‘)
   {
        printf(“\n Voce pressionou p”);
        printf(“\n Pressione qualquer tecla para terminar”);
        getchar( );
    }
}
```

5.1.2 - Comando IF - ELSE

- Determina duas ações: Uma para quando a expressão do teste for verdadeira e outra para quando for falsa.

Sintaxe:

```
if   (<expressão>)
     <comando1(s)>;
else
     <comando2(s)>;
```

Ex.1:

```
main ( )
{
  char ch;
  ch = getchar( );
  if  (ch == ‘p’)
       printf(“\n Você pressionou p”);
  else
      printf (“\n Você não pressionou p”);
}
```

- Podemos ter comandos aninhados:

Ex. 2:

```
if ( n>0 )
        if ( a>b )
                z = a;
        else
           z = b;
```

obs.: O else sempre pertence ao *if* mais interno, em caso de dúvida utilize { }.

Ex. 2:

```
if ( n>0 )
{
        if ( a>b )
                z = a;
        else
                z = b;

}
```

```
if  (n > 0)
{
      if  (a>b)
          z = a;
 }
 else

      z = b;
```

Ex. 3:

Seja o seguinte trecho de programa:

```
if ( a == 1)
        inc1 ++ ;
else   if (a == 2)
                inc2 ++;
        else   if ( a == 3)
                        inc3 ++;
                else   inc ++;
```

Todas as decisões acima dependem da variável **a**. Logo este trecho pode ser melhor escrito, usando o comando *switch*. (case no Pascal)

5.1.3 - Comando SWITCH_

- É um comando de decisão múltipla. Verifica se uma expressão assume algum valor relacionado e executa ações de acordo com o resultado.

Sintaxe:

```
switch  < expressão >
        {
        case valor1 : <comando(s)>;
        case valor2 : <comando(s)>;
                .
                .
                .
        default : <comando(s)>;
        }
```

- O caso *_default* é opcional, seus comandos são executados quando nenhum outro caso é satisfeito.

O ex.3 do item 4.1.2 pode ser rescrito da seguinte maneira:

Ex.1.

```
Switch (a) {
        case  1  :  inc1 ++;  break;
        case  2  :  inc2 ++;  break;
        case  3  :  inc3 ++;  break;
        default  :  inc  ++;  break;


        }
```

- O comando **break** causa saída imediata do *switch*. Se não for usado o *break*, o programa continuará executando os outros comandos situados abaixo.

Ex. 2.

Seja o seguinte trecho de programa:

```
if ( ( a == 1 ) || ( a == 2 ) || ( a = =3 ) )
        inc123 ++;
else   if ( a  = = 4 )
                inc4 ++;
        else   if ( a = = 5)
                        inc5 ++;
```

O mesmo poderia ser rescrito como:

```
switch (a)  {
                case 1:
                case 2:
                case 3: inc123 ++; break;
                case 4: inc4 ++; break;
                case 5: inc5 ++; break;
                }
```

5.2 - Comandos de iteração (laços)

Controlam a repetição de um ou mais comandos. São eles: *for*, *while* e *do-while.*

Exercício, fazer um programa Pascal que imprima os números inteiros de 0 a 9.

5.2.1 - Comando FOR

- É útil para repetir algo um número fixo de vezes.

Sintaxe:

for (<inicialização> ; <teste> ; <incremento>)

<comando(s)>;

- Contém três expressões separadas por " ; "

Ex. 1.

```
#include <stdio.h>
main( )
{
int i;
for (i = 0 ; i < 10 ; i ++ )
        printf ("conta = %d \n", i);
}
```

Qual a saída do programa acima?

Resposta: números inteiros de 0 a 9.

- Qualquer uma das expressões de um *for* pode conter várias instruções separadas por vírgulas.

```
#include <stdio.h>
main( )
{
  int i;
  for (i = 0 ; i < 10 ;  printf ("conta = %d \n", i), i ++ );
}
```

```
for (i = 0 ; i < 10 ;  printf ("conta = %d \n", i ++ ));  /* 0 .. 9*/

for (i = 0 ; i < 10 ;  printf ("conta = %d \n", ++i));   /* 1 .. 10*/
```

Ex. 2.

```
 main( )
{
int x, y;
for (x = 0, y = 0  ;  x + y < 100  ;  x ++,  y ++ )
        printf ( "%d ",  x + y );
}
```

Qual a saída do programa ???

Resposta: 0, 2, 4, 6, 8, ..., 98.

- Podemos usar caracteres ao invés de inteiros:

Ex. 3.

```
main ( )
{
char ch;
for ( ch = 'a' ;  ch< = 'z' ;  ch ++ )
        printf ("VALOR ASCII de % c  é  %d \n", ch, ch);
}
```

- É possível fazer chamadas às funções, em qualquer uma das expressões do laço:

Ex. 4.

```
main ( )
{   char ch;
for  ( ch = getchar( ) ;  ch != 'x' ;  ch = getchar( )  )
        printf ( "%c ", ch + 1 );
}
```

O que faz este programa?

Resposta: Imprime o próximo da tabela em forma de caracter.

- Pode-se omitir qualquer uma das três partes de um for , no entanto os ponto-e-vírgulas devem permanecer.

Ex. 5.
```
main( )
{
 char ch;
for  ( ;  ( ch = getchar ( ) ) != 'x' ;  )
        printf ( "%c " ,  ch + 1 );
}
```

Obs.: Não tem valor inicial e nem incremento, mas no teste ele está lendo e verificando se continua ou não. Se continuar, imprime o próximo caracter.

Ex. 6.
```
main( )
{   for ( ;  ; )  printf ("Laço Infinito \n");
}
```

Obs.: A instrução for (; ;) faz com que o laço seja infinito.

Ex. 7.
```
main( )
{   char ch ;
    for ( ; (ch = getchar( ) ) != 'x' ;  printf ("% c ", ch + 1);
}
```

Obs.: Não tem valor inicial, lê ch , testa condição para continuar, não executa incremento e sim outro comando e não apresenta comandos subordinados ao for .

- Podemos ter instruções múltiplas na inicialização:

Ex. 8.

```
main( )
{
int conta, total;
for ( conta = 0, total = 0 ; conta < 10  ;  conta ++ )
        {
          total + = conta;  /* total = total +conta */
          printf (“conta = % d , total = % d \n “, conta, total );
        }
}
```

Obs.: **total** foi inicializado no *for*, mesmo sem participar do teste e do incremento.

- Podemos ter laços *for* aninhados:

Ex. 9

```
k = 0;
for ( i = 0 ; i < m ; i ++ )
        for ( j = 0 ; j < n ; j ++ )
                if ( A[i] = = B[j] )
                        k ++ ;
```

O que este trecho de programa faz?

Resposta: Compara cada elemento de A com todos os elementos de B e conta o número de elementos repetidos em A e B.

5.2.2 - Comando WHILE

- Utiliza os mesmos elementos do comando *for*, porém de maneira diferente.

Sintaxe:

```
while ( < expressão > )
        <  comando(s) > ;
```

Ex. 1.
Outra versão do Ex. 8 do item 4.2.1:

```
main( )
{
  int conta = 0 , total = 0 ;
 while  ( conta < 10 )
 {
   total + = conta ;
   printf ("conta = % d , total = % d \n ", conta ++ , total);
 }
}
```

- Deve-se usar *while* em situações onde não se sabe quantas vezes um comando precisa ser executado.

Ex. 2.

```
# include <stdio.h>
main( )
{   int cont = 0;
    printf ( "Digite uma frase:   \n");
    while  (getchar ( ) ! = '\r' )     /* caracter ≠ <enter> */
          cont ++;
     printf ( "\n O número de caracteres é % d ", cont );
}
```

- Podemos ter laços *while* aninhados.

Ex. 3.

Outra versão do Ex. 9 anterior, usando *while*:

```
k = 0;
i = 0;
while ( i < m )
        {
         j = 0;
         while (j < n )
                if ( A[i] = = B[j + +] )
                        k + +;
                i + +;
        }
```

5.2.3 - Comando DO-WHILE

- Executa um ou mais comandos. Se o teste for verdadeiro, executa novamente os comandos. O teste é feito no final da execução do bloco.

Sintaxe:

```
do
     {  <comando(s) >
      }
while  ( < expressão > );        /* ≠  0 */
```

Ex.:

Outra versão do Ex. 9 do item 4.2.1 (*for*):

```
k=i=0;
do    {
      j=0;
      do {
           if ( A[ i ]) = = B[ j ])
             k + +;
           j + +;
           }
      while  (j < n)
             i + +;
      }
while (i < m);
```

- *do-while* é menos usado do que *for* e *while*.
- Legibilidade : Se o teste estiver no início, é mais fácil entender o objetivo do laço. No comando *do-while*, se o teste for falso de início, os comandos serão executados pelo menos uma vez.

5.3 - Comandos de desvio

Realizam um desvio incondicional, ou seja, não testam condições.
São eles: *goto*, *return*, *break* e *continue*.

5.3.1 - Comando GOTO

- Executa um desvio para a localização do programa onde está o rótulo.

Sintaxe:

```
goto  < rótulo >;
        .
        .
        .
< rótulo > : < comando(s) >;
```

O rótulo deve estar em alguma parte do programa.

- Programas com *goto* são menos legíveis e são confusos. É fácil escrever programas sem *goto*.

5.3.2 - Comando BREAK

- É utilizado em conjunto com o comando *switch*, conforme visto anteriormente.

- Quando usado dentro de um laço, causa a saída imediatamente do mesmo e vai para a próxima parte do programa:

Ex.:

```
for ( i = 0 ; i < n ; i ++ )
        if   ( A[i] = = x )
            break;
```

5.3.3 - Comando CONTINUE

Ao ser utilizado dentro de um laço, pula para a próxima iteração, não executando os comandos restantes.

Ex.1:

```
x = 0;
for ( i = 0 ; i < n ; i + + )
     {   x += A[i]  ;
       i ++;
       if (A[i] ==0)
          continue;
       z *= A[i];        /* não  será executado se A[i] for 0 */
      }
```

5.3.4 - Comando RETURN

- Serve para retornar de uma função, isto é, faz a execução “saltar” ao ponto onde a função foi chamada.
- Pode ter um valor associado que é o valor de retorno da função.

Sintaxe:

return(<expressão>);

Ex.:

return(x);

return( );

5.3.5 – Comando EXIT

- Encerra a aplicação. O status informado como parâmetro é devolvido ao sistema operacional (ao shell que estiver rodando a aplicação)
- Função da biblioteca stdlib.h.

Sintaxe:

exit(<status>);

Ex.: exit (0);