Métodos Computacionais

Comandos Condicionais e de Repetição em C

Tópicos da Aula

- Hoje vamos acrescentar comportamentos mais complexos a programas em C
  - Comandos Condicionais
    - *if-else*
    - *switch*
  - Comandos de Repetição
    - for
    - while
    - do-while
  - Comandos de Desvio
    - break
    - continue

Comandos Condicionais

- Um **comando condicional** nos permite escolher qual deve ser a próxima instrução executada em um programa
- A execução de uma determinada instrução depende de uma condição (expressão booleana)
- A linguagem C oferece 3 tipos de comandos condicionais:
  - if – else
  - switch
  - comando ternário (operador condicional)

O comando if-else

```
if (expressaoBooleana) {
   comandos
} else {
   outros comandos
}
```

Se a avaliação de **expressaoBooleana** retornar **verdadeiro**, **comandos** são executados , caso contrário, executa-se **outros comandos**

Exemplo do if-else

```
#include   <stdio.h>
 int main ( )
{ float  n1, n2, n3, m;
   printf (“\nEntre com 3 notas ” ) ;
   scanf (“%f %f %f”, &n1, &n2, &n3);
   m  =  (n1 + n2 + n3 ) / 3 ;
   if (m >= 7.0)   {
      printf (“\n Aluno aprovado. ”) ;
      printf (“ Média igual a %f “ , m) ;
  }else  {
      printf (“\n Aluno reprovado. ”);
      printf (“ Média igual a %f “ , m) ;
  }
  return 0;
}
```

Variações do comando if-else

```
if (expressaoBooleana) {
   comandos
}
```

```
if (expressaoBooleana)
   comando;
```

Se a avaliação da expressão retornar **falso**, não executa-se nada

```
if (expressaoBooleana)
   comando;
else outroComando;
```

O uso do bloco só é necessário caso queira-se executar mais de um comando

Exemplo – if sem else

```
#include    <stdio.h>
 int main ( )
 { int   resposta ;
   printf (“\n Qual o valor de  10 + 14? ”);
   scanf (“%d”, &resposta);
   if  (resposta == 10 + 14)
       printf (“\n Resposta correta ! ”);
   return 0;
}
```

else não é obrigatório

Exemplo – if-else com único comando

```
#include    <stdio.h>
 int main ( )
 { int   num ;
   printf (“\nDigite um número:   ”);
   scanf (“%d”, &num);
   if  (num <  0)
       printf (“\n Número é negativo ! ”);
   else
       printf (“\n Número é positivo ! ”);
    return 0;
  }
```

Aninhando if-else

```
if (expressaoBooleana)
   if (expressaoBooleana)
      comando;
   else outroComando;
```

if-else aninhado

O comando dentro do if ou else pode ser outro if

Num aninhamento, o else é associado ao mais recente if sem else

Exemplo if-else Aninhado

- Exercício: O que está errado neste programa?

```
#include   <stdio.h >
 int main ()
{ int  temp ;
   printf (“\n Digite a temperatura: ” ) ;
   scanf (“%d”, &temp) ;
   if (temp < 30)
      if (temp  > 20)
         printf (“\n Temperatura agradável ” ) ;
   else printf (“\n Temperatura muito quente ” ) ;
   return  0 ;
 }
```

Exemplo if-else Aninhado

- Exercício: Corrigindo o programa

```
#include   <stdio.h >
 int main ()
{ int  temp ;
   printf (“\n Digite a temperatura: ” ) ;
   scanf (“%d”, &temp) ;
   if (temp < 30) {
      if (temp  > 20)
         printf (“\n Temperatura agradável ” ) ;
   }else printf (“\n Temperatura muito quente ” ) ;
   return  0 ;
 }
```

else associado ao **if** certo

Encadeando comandos if-else

```
if (expressaoBooleana) {
   comandos
} else if (expressaoBooleana') {
   comandos'
} else {
   comandos''
}
```

Tomando Múltiplas Decisões

- O comando if-else é útil para a escolha de uma entre duas alternativas
- Quando mais de duas alternativas são necessárias, pode ficar deselegante utilizar vários if-else encadeados
  - Para estes casos o comando switch pode ser a melhor opção

O Comando switch

```
switch(expressao) {
   case rotulo1:
      Comandos1
      break;
   case rotulo2:
      Comandos2
      break;
   ...
   default:
      Comandos
}
```

Para executar um `switch`

- Avalia-se **`expressao`**
- Executa-se os comandos do **`case`** cujo rótulo é igual ao valor resultante da expressão
- Executa-se os comandos de **`default`** caso o valor resultante não seja igual a nenhum rótulo

Restrições do Comando switch

```
switch(expressao) {
  case rotulo1:
    Comandos1
    break;
  case rotulo2:
    Comandos2
    break;
  ...
  default:
    Comandos
}
```

- O tipo de **expressao** só pode ser :
  - Inteiro ou caractere
- Os rótulos são constantes diferentes
- Existe no máximo uma cláusula **default** (é opcional)
- Os tipos dos rótulos têm que ser o mesmo de **expressao**

Variações do Comando switch

```
switch(expressao) {
  case rotulo1:
    Comandos1
    break;
  case rotulo2:
    Comandos2
    break;
  ...
  default:
    Comandos
}
```

- Vários rótulos podem estar associados ao mesmo comando
- Os comandos **break** são opcionais:
  - Sem o **break** a execução dos comandos de um rótulo continua nos comandos do próximo, até chegar ao final ou a um **break**

Exemplo de switch

- Calcular a diferença, o produto, o quociente ou a soma de dois números, dependendo da operação escolhida e imprimir o resultado.

```
# include    “stdio.h“
 int main (void )
 {
 char x;
 int a, b ;
 float result = 0.0;
 printf (“\n Informe os 2 números e a operação ”);
 scanf (“%d %d %c”, &a, &b, &x);
```

Exemplo de switch (cont..)

```
switch (x) {
   case  '+':   result = a + b;
                break;
   case  '-':   result = a - b;
                break;
   case  '*':   result = a * b;
                break;
   case  '/':   result = a / b;
                break;
   default :  printf("\nOperador invalido");
}
printf ("\nResultado igual a %f ", result);
return 0 ;
}
```

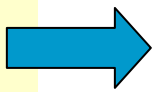
Operador Condicional - ?

- Forma Geral do ?

```
condição ?  expressão1  :  expressão2
```

- Substitui construções do tipo:

```
if ( a > b )  {
      maximo =  a;
} else  {
      maximo = b;
}
```



```
maximo = a > b ? a :b ;
```

Comando Ternário de Decisão - ?

- Exercício: Considere as seguintes declarações

```
char a = 'a', b = 'b';  /*b tem valor 98*/
int  i = 1,  j = 2 ;
double  x = 7.07 ;
```

Expressão	Valor
`i == j ? a-1: b + 1`	'c'
`j % 3 == 0 ? i + 4 : x`	7.07
`j > 1 ?  j - 1 : j + 1`	1
`j % 3 != 0 ? i + 4:  x`	5

Estruturas de Repetição

- Permite repetir diversas vezes um comando ou seqüência de comandos
  - Cada repetição de um comando ou seqüência de comandos é denominada de **iteração**
- São geralmente conhecidos como *loops*(laços)
- Da mesma forma que comandos condicionais, são controladas por expressões *booleanas*
- C oferece 3 tipos de estruturas(comandos) de repetição:
  - O laço *for*
  - O laço *while*
  - O laço *do-while*

O Comando for

```
for (i = 0; i < valor; i = i+1)
  corpo
```

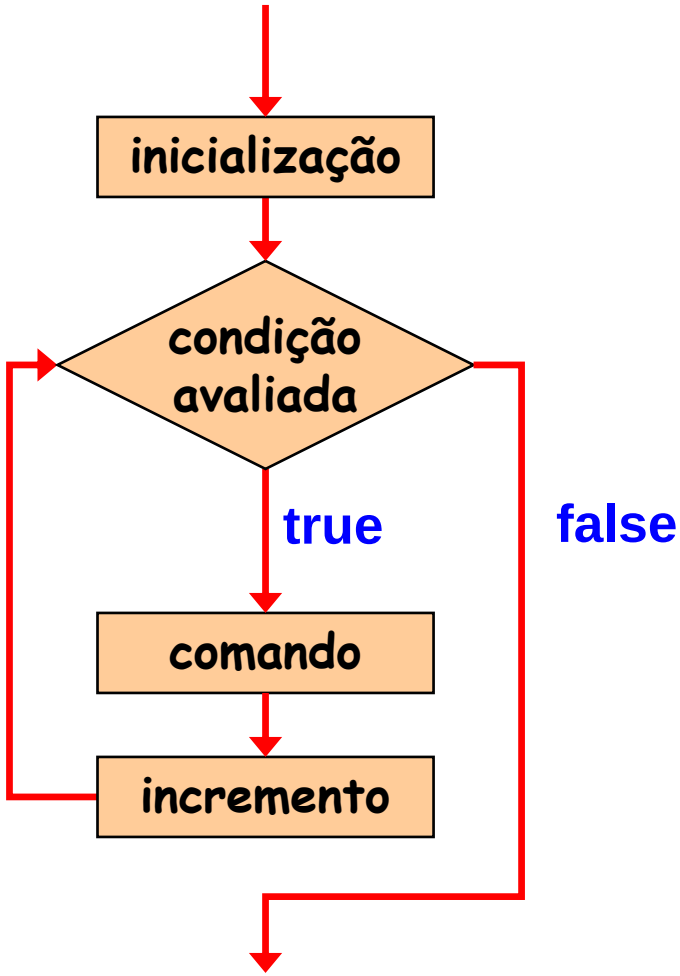
- Executa **corpo** um número específico de vezes: **valor** vezes
- Neste exemplo, na primeira execução de **corpo**, o valor de **i** é **0**
- O valor de **i** é incrementado após cada execução de **corpo**
- Variável **i deve ser declarada** antes de se utilizar o comando for
  - **Variável de controle**

A Forma Geral do Comando for

```
for(inicialização;condição;incremento)
  corpo
```

- **inicialização** e **incremento** podem ser praticamente quaisquer comandos
- **condição** pode ser qualquer expressão booleana
- **Inicialização** geralmente inicializa a variável de controle do **for**
- **incremento** geralmente incrementa a variável **for**

Fluxo de Controle do Laço for



Examinando o Comando *for*

A ***inicialização*** é executada uma só vez antes do laço começar

O ***comando*** é executado até que ***condição*** se tornar *falsa*

```
for ( inicialização ; condição ; incremento )
    comando;
```

O ***incremento*** é executado ao fim de cada iteração

Cabeçalho do for

Entendendo o Comando for

```
int somatorio(int n) {
      int valor;
      int soma = 0;
      for (valor = 0; valor <= n; valor++)
            soma = soma + valor;
      return soma;
}
```

Variável **valor** é inicializada com **0**

Comando será realizado enquanto **valor** for menor ou igual a **n**

A cada iteração, **valor** é incrementado em 1

Entendendo o Comando for

```
int somatorio(int n) {
      int valor;
      int soma = 0;
      for (valor = 0; valor <= n; valor++)
            soma = soma + valor;
      return soma;
}
```

Se **n** for menor do que **0**, não se executa o corpo do **for**

É executado depois do **for**

Entendendo o Comando for

```
int somatorio(int n) {
      int soma = 0;
      int valor;
      for (valor = 0; valor <= n; valor++){
            soma = soma + valor;
            printf("Soma Parcial:%d\n",soma);
      }
      printf("Soma Total:%d",soma);
      return soma;
}
```

Corpo do **for** pode ser composto por bloco de comandos

Modificando o Incremento do for

```
int somatorio(int n) {
      int soma = 0;
      int valor;
      for (valor = n; valor>= 0; valor--){
            soma = soma + valor;
            printf("Soma Parcial:%d\n",soma);
      }
      printf("Soma Total:%d",soma);
      return soma;
}
```

valor agora é decrementado

Modificando o Incremento do for

```
int somatorioPares(int n) {
      int soma = 0;
      int valor;
      for (valor = 0; valor <= n; valor = valor + 2){
            soma = soma + valor;
            printf("Soma Parcial:%d\n",soma);
      }
      printf("Soma Total:%d",soma)
      return soma;
}
```

valor agora é incrementado em 2

- Pode-se colocar qualquer tipo de expressão na parte de incremento do comando *for*

Variações do Comando for

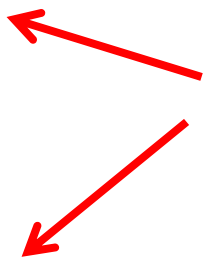
- Cada expressão no cabeçalho de um laço `for` loop é opcional
- Se a inicialização é omitida, nenhuma inicialização é feita
- Se a condição é omitida, a condição é considerada sempre *verdadeira* e o laço continua para sempre (laço infinito)
- Se o incremento é omitido, nenhuma operação é realizada ao final da iteração do laço

O Comando for sem Condição, etc.

```
for(;valor < n; valor++)
   corpo
```

```
for(;;valor++)
   corpo
```

```
for(;;)
   corpo
```



Repetição infinita:
cuidado!

Mais Variações do Comando *for*

- Teste condicional não precisa ser baseado na variável de controle

```
#include   <stdio.h>
#include   <conio.h>
 int main ( ) {
    int  i ;
   /* atribui um valor inicial a ch*/
    char  ch  =  'a' ;
    for(i = 0; ch != 'q';i++){
        printf ("\nPasso:%d",i) ;
        ch = getche() ;
    }
}
```

condição não é baseada na variável de controle

Mais Variações do Comando *for*

- Parte de incremento não precisa incrementar variável de controle
- Comando for pode vir sem corpo

```
#include <stdio.h>
#include <conio.h>
int main ( ) {
  char ch;
  for(ch = getche(); ch!='q';ch = getche());
  printf ("\nValor q encontrado") ;
}
```

Não tem corpo

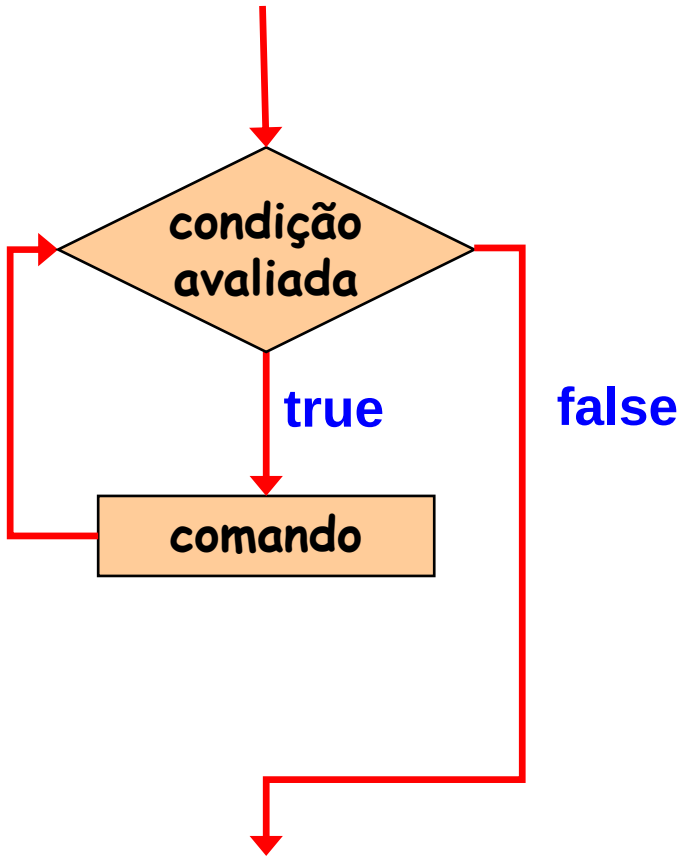
Incremento pode ser outro tipo de expressão

O comando while

```
while (condição)
   corpo
```

- Executa **corpo** várias vezes até que a avaliação da expressão retorne **falso**
- A condição é avaliada de novo após cada execução de **corpo**
- Não executa **corpo** nenhuma vez, se de início a avaliação da condição retorna **falso**

Fluxo de Controle do Laço `while`



Entendendo o comando while

```
int somatorio(int n) {
      int soma = 0;
      int valor = 0;
      while ( valor <= n ){
            soma = soma + valor;
            valor++;
      }
      return soma;
}
```

Se **n** for negativo, não se executa o corpo do **while**

É executado quando o **while** termina, quando a condição for **falsa**

Entendendo o comando while

```
int somatorio(int n) {
      int soma = 0;
      int valor = 0;
      while ( valor <= n ){
            soma = soma + valor;
            valor++;
      }
      return soma;
}
```

Inicialização da variável de controle é feita fora do laço **while**

Incremento da variável de controle é feita no corpo do laço **while**

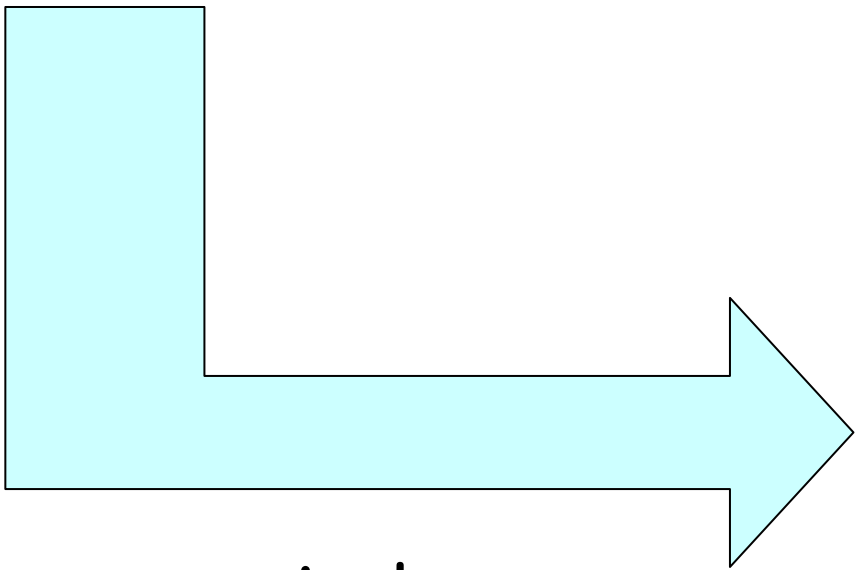
Laço Infinito com o Comando while

```
int somatorio(int n) {
      int soma = 0;
      int valor = 0;
      while ( valor <= n ){
            soma = soma + valor;
      }
      return soma;
}
```

Se valor não é incrementado, este comando será executado infinitas vezes

O Comando for e o Comando while

```
for(inicialização;condição;incremento)
   corpo
```

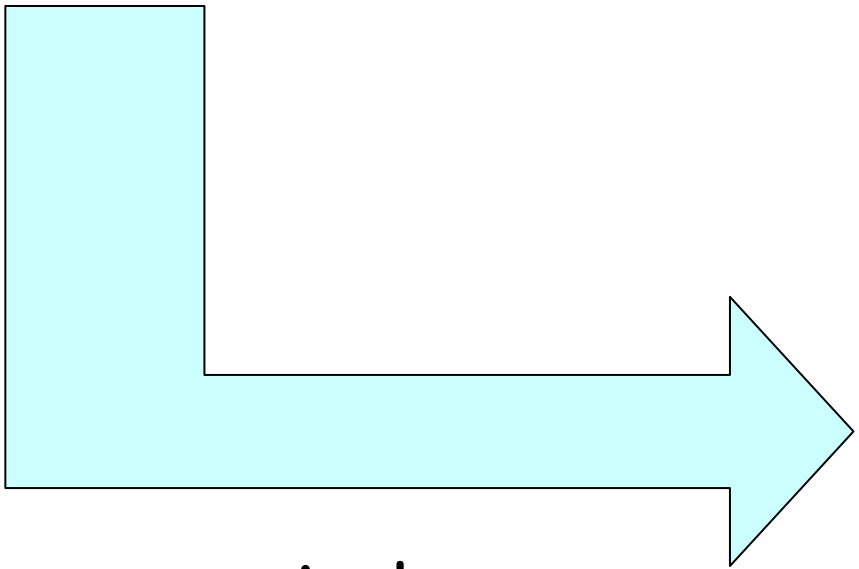


equivale a ...

```
inicialização;
while(condição) {
   corpo;
   incremento;
}
```

O Comando for e o Comando while

```
for(;;)
   corpo
```



equivale a ...

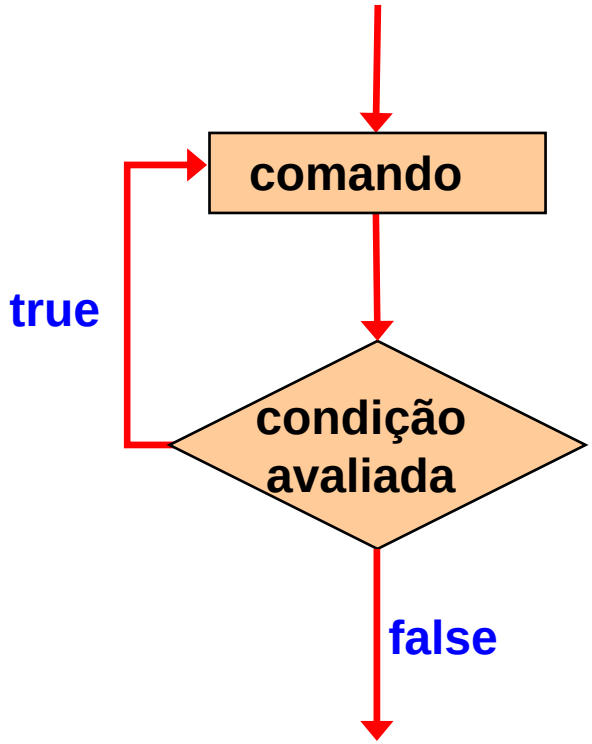
```
while(1) {
   corpo;
}
```

O Comando do-while

```
do {
   corpo
} while(condição)
```

- Executa **corpo,** pelo menos uma vez, até que a avaliação da condição retorne **falso**
- A condição é avaliada de novo após cada execução de **corpo**

Fluxo de Controle do Laço do-while



Entendendo o comando do-while

```
int somatorio(int n) {
      int soma = 0;
      int valor = 0;
      do {
            soma = soma + valor;
            valor++;
      } while ( valor <= n )

      return soma;
}
```

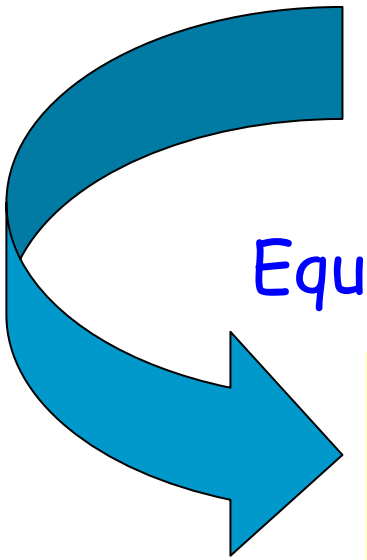
Se **n** for negativo, o corpo do **do-while** é executado pelo menos uma vez

É executado quando o **do-while** termina, quando a condição for **falsa**

Comportamento alterado: cuidado!

Os Comandos do-while e while

```
do {
   corpo
} while(condição)
```



Equivalente a ...

```
corpo;
while(condição)
   corpo;
```

Laços Aninhados

- Laços podem ser aninhados da mesma forma que comandos condicionais
  - O corpo de um laço pode conter outro laço
- Para cada iteração do laço externo, o laço interno é completamente executado

Laços Aninhados

```
int somatorioDoSomatorio(int n, int vezes) {
      int soma = 0, somatorio =0;
      int valExt;
      for (valExt = 0; valExt < vezes; valExt++ ){
            int valInt = 0;
            while (valInt <= n) {
                  soma = soma + valInt;
                  valInt++;
            }
            somatorio = somatorio + soma;
      }
      return somatorio;
}
```

A cada iteração do **for**, o laço **while** é executado

Considerações sobre Laços

- Os 3 tipos de laços são funcionalmente equivalentes
  - Portanto podem ser usados indiscriminadamente
- Os laços **`for`** e **`while`** são executados 0 ou muitas vezes
- O laço **`do-while`** é executado 1 ou muitas vezes

O Comando `break`

- Forma Geral do comando **`break`**

```
break;
```

- Tem dois usos distintos
  - Para forçar o término de um laço de repetição (*do-while*, *for* ou *while*)
  - Para terminar um *case* do comando *switch*
- Quando o comando **`break`** é encontrado dentro de um laço de repetição:
  - instrução é imediatamente finalizada
  - próxima instrução após a estrutura de repetição é executada
- Deve ser usado com cautela
  - Reduz legibilidade
  - Pode levar a erros de lógica

O Comando break

Este comando não será executado quando valor = = n

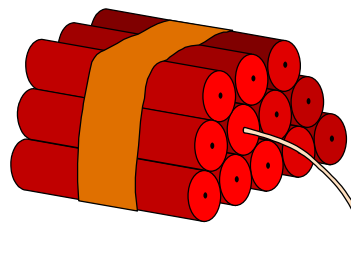
```
int somatorio(int n) {
      int soma = 0;
      int valor;
      for (valor = 0; ; valor++){
            soma = soma + valor;
            if (valor == n)
                  break;
            printf("Soma Parcial:%d\n",soma);
      }
      printf("Soma Total:%d",soma)
      return soma;
}
```

O Comando continue

- Forma Geral do comando **continue**

```
continue
```

- Termina a execução da iteração atual de um loop (**for**, **while**, **do-while**) e volta ao começo deste loop
- Todos os comandos que seriam executados após o **continue** são descartados



Comando de Desvio - continue

- Para os comandos *while* e *do-while*, o **`continue`** causa:
  - a realização imediata do teste da condição correspondente
  - continuidade do processo de repetição dependendo do resultado do teste
- Para o comando *for* , o **`continue`** causa:
  - incremento da variável de controle do laço de repetição
  - execução do teste para verificação da condição
  - continuidade do processo de repetição dependendo do resultado do teste

O Comando continue

```
void imprimeNumerosAteCinco() {
     int valor;
     for (valor = 0; valor <= 5; valor++){
          if (valor == 4)
               continue;
          printf("%d ",valor);
     }

}
```

Controle pula para o incremento e este comando não será executado quando **valor = = 4**

A saída desta função será: **0 1 2 3 5**

O Comando continue

```
void imprimeNumerosAteCinco() {
      int valor = 0;
      while (valor <= 5){
            if (valor == 4)
                  continue;
            printf("%d",valor);
            valor++;
      }

}
```

Controle pula para o teste e função tem **execução infinita** quando **valor = = 4**

A saída desta função será: 0 1 2 3 ...

laço infinito