

Tema: Introdução à programação

Atividade: Testes, repetições e alternativas em C

- 01.) Editar e salvar um esboço de programa em C, com o nome do arquivo Exemplo0200.c  
(não usar espaços em branco em nomes de pastas ou arquivos), observar o uso de pontuação, maiúsculas e minúsculas, espaços em branco entre operações e não usar acentos ou cedilha:

```
/*
Exemplo0200 - v0.0. - __ / __ / ____
Author: _____

Para compilar em terminal (janela de comandos):
Linux   : gcc -o exemplo0200    exemplo0200.c
Windows: gcc -o exemplo0200    exemplo0200.c

Para executar em terminal (janela de comandos):
Linux   : ./exemplo0200
Windows: exemplo0200
*/
// dependencias
#include "io.h"          // bibliotecas e outras definicoes

/**
Method_01.
*/
void method_01 ( void )
{
    // identificar
    IO_id ( "Method_01 - Programa - v0.0" );

    // encerrar
    IO_pause ( "\nApertar ENTER para continuar.\n" );
} // end method_01 ( )

/*
Funcao principal.
@return codigo de encerramento
*/
int main ( void )
{
    // definir dado
    int opcao = 0

    // identificar
    printf ( "%s\n", "Exemplo0200 - Programa = v0.0" );
    printf ( "%s\n", "Autor: _____" );
    printf ( "\n" );          // mudar de linha
```

```

// acoes

// repetir
do
{
    // para mostrar opcoes
    printf ( "\n%s\n", "Opcoes:" );
    printf ( "\n%s" , "0 - Terminar" );
    printf ( "\n%s" , "1 - Method_01" );
    printf ( "\n" );

    // ler a opcao do teclado
    printf ( "\n%s", "Opcao = " );
    scanf ( "%d", &opcao );
    getchar(); // para limpar a entrada de dados

    // para mostrar a opcao lida
    printf ( "\n%s%d", "Opcao = ", opcao );

    // escolher acao dependente da opcao
    switch ( opcao )
    {
        case 0: /* nao fazer nada */ break;
        case 1: method_01 ( ); break;
        default: // comportamento padrao
            printf ( "\nERRO: Opcao invalida.\n" );
            break;
    } // end switch
}
while ( opcao != 0 );

// encerrar
printf ( "\n\nApertar ENTER para terminar." );
getchar(); // aguardar por ENTER
return ( 0 ); // voltar ao SO (sem erros)
} // end main ( )

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

----- historico

Versao      Data      Modificacao
0.1         __/__/__   esboco

----- testes

Versao      Teste
0.0         00. ( __ )   identificacao de programa

*/

```

02.) Compilar o programa.

Se houver erros, identificar, individualmente, a referência para a linha onde ocorre.

Consultar atentamente o modelo acima, na linha onde ocorreu o erro (e também linhas próximas), editar as modificações necessárias.

Compilar novamente e proceder assim até que todos os erros tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

DICA: Se precisar de ajuda sobre como proceder a compilação, consultar os vídeos com as demonstrações sobre algumas formas para fazê-lo.

SUGESTÃO: Para se acostumar ao tratamento de erros, registrar a mensagem de erro (como comentário) e quais as medidas encontradas para resolvê-lo.

03.) Executar o programa.

Observar as saídas.

04.) Editar e salvar um esboço de programa em C,

com as seguintes modificações para usar alternativa simples:

```
/**
 *Method_01.
 */
void method_01 ( void )
{
    // definir dado
    int x = 0;           // definir variavel com valor inicial

    // identificar
    IO_id ( "Method_01 - Programa - v0.0" );

    // ler do teclado
    x = IO_readint ( "Entrar com um valor inteiro: " );

    // testar valor
    if ( x == 0 )
    {
        IO_printf ( "%s (%d)\n", "Valor igual a zero", x );
    }
    if ( x != 0 )
    {
        IO_printf ( "%s (%d)\n", "Valor diferente de zero ", x );
    } // end if

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // end method_01 ( )
```

/\*

----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

a.) 0

b.) 5

c.) -5

----- historico

Versao	Data	Modificacao
0.1	__/__/__	esboco

----- testes

Versao	Teste	
0.1	01. ( OK )	identificacao de programa

\*/

DICA: O uso de blocos { } é facultativo no caso de haver apenas um comando.  
Recomenda-se, no entanto, fazer o uso de blocos mesmo nesse caso,  
para facilitar a depuração e a correção de programas.

05.) Compilar o programa.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

Em caso de dúvidas, consultar a apostila, recorrer aos monitores ou apresentá-las ao professor.

06.) Executar o programa.

Observar as saídas.

Registrar os resultados.

Em caso de erro (ou dúvida), usar comentários para registrar a ocorrência e,  
posteriormente, tentar resolvê-lo (ou esclarecer a dúvida).

07.) Incluir outro método para usar uma alternativa dupla.

Prever novos testes.

Sugestão: Se quiser realizar controle de versão, alterar o nome do programa.

```

/*
  Method_02.
*/
void method_02 ( void )
{
  // definir dado
  int x = 0;          // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_02 - Programa - v0.0" );

  // ler do teclado
  x = IO_readint ( "Entrar com um valor inteiro: " );

  // testar valor
  if ( x == 0 )
  {
    IO_printf ( "%s (%d)\n", "Valor igual a zero", x );
  }
  else      // equivalente a "caso diferente do já testado"
  {
    IO_printf ( "%s (%d)\n", "Valor diferente de zero ", x );
  } // end if

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method_02 ( )

/*
----- documentacao complementar
----- notas / observacoes / comentarios
----- previsao de testes

a.) 0
b.) 5
c.) -5

----- historico

Versao    Data        Modificacao
0.1       __/___      esboco

----- testes

Versao    Teste
0.1       01. ( OK )   identificacao de programa
*/

```

08.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

09.) Executar o programa.  
Observar as saídas.  
Registrar os resultados.

10.) Incluir outro método para usar uma alternativa dupla, aninhada.  
Prever novos testes.

```
/*
  Method_03.
*/
void method_03 ( void )
{
  // definir dado
  int x = 0;          // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_03 - Programa - v0.0" );

  // ler do teclado
  x = IO_readint ( "Entrar com um valor inteiro: " );

  // testar valor
  if ( x == 0 )
  {
    IO_printf ( "%s (%d)\n", "Valor igual a zero", x );
  }
  else
  {
    IO_printf ( "%s (%d)\n", "Valor diferente de zero ", x );
    if ( x > 0 )
    {
      IO_printf ( "%s (%d)\n", "Valor maior que zero", x );
    }
    else
    {
      IO_printf ( "%s (%d)\n", "Valor menor que zero", x );
    } // end if
  } // end if

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method_03 ( )
```

```

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

```

- a.) 0
- b.) 5
- c.) -5

```

----- historico

```

Versao	Data	Modificacao
0.1	__/__/__	esboco

```

----- testes

```

Versao	Teste	identificacao de programa
0.1	01. ( OK )	

```

*/

```

DICA: O uso da alternativa (**else**) é facultativo.  
 Recomenda-se, no entanto, fazer o uso do bloco alternativo,  
 para evitar o uso de outro com a negação do teste já realizado (**if**).  
 Minimiza-se o custo computacional ao realizar apenas um teste ao invés de dois.

- 11.) Compilar o programa novamente.  
 Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.  
 Se não houver erros, seguir para o próximo passo.
- 12.) Executar o programa.  
 Observar as saídas.  
 Registrar os resultados.
- 13.) Incluir outro método para usar condições compostas nos testes.  
 Prever novos testes.

```

/*
  Method_04.
*/
void method_04 ( void )
{
  // definir dado
  double x = 0.0;      // definir variavel com valor inicial

  // identificar
  IO_id ( "EXEMPLO0204 - Programa - v0.0" );

  // ler do teclado
  x = IO_readdouble ( "Entrar com um valor real: " );

```

```
// testar valor
if ( 1.0 <= x && x <= 10.0 )
{
    IO_printf ( "%s (%lf)\n", "Valor dentro do intervalo [1:10] ", x );
}
else
{
    IO_printf ( "%s (%lf)\n", "Valor fora do intervalo [1:10] ", x );
    if ( x < 1.0 )
    {
        IO_printf ( "%s (%lf)\n", "Valor abaixo do intervalo [1:10] ", x );
    }
    else
    {
        IO_printf ( "%s (%lf)\n", "Valor acima do intervalo [1:10]", x );
    } // end if
} // end if

// encerrar
IO_pause ( "Apertar ENTER para continuar" );
} // end method_04 ( )
```

```
/*
----- documentacao complementar
----- notas / observacoes / comentarios
----- previsao de testes
```

- a.) 0
- b.) 1
- c.) 10
- d.) -1
- e.) 100

```
----- historico
```

Versao	Data	Modificacao
0.1	__/__/__	esboco

```
----- testes
```

Versao	Teste	
0.1	01. ( OK )	identificacao de programa

```
*/
```

DICA: O uso da conjunção lógica (&&) é necessário para avaliar o pertencimento ao intervalo. O bloco alternativo (else) será acionado caso não houver pertencimento ao intervalo. Entretanto, como pode há duas condições para que isso possa acontecer (abaixo ou acima), será necessário realizar outro teste para se distinguir entre essas. Recomenda-se o uso da endentação dos blocos a fim de se proporcionar melhor identificação da vinculação entre condições.

14.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.



- 15.) Executar o programa.  
Observar as saídas.  
Registrar os resultados.
- 16.) Incluir outro método para acrescentar várias condições compostas e aninhadas.  
Prever novos testes.

```
/*
  Method_05.
*/
void method_05 ( void )
{
  // definir dado
  char x = '_';          // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_05 - Programa - v0.0" );

  // ler do teclado
  x = IO_readchar ( "Entrar com um caractere: " );

  // testar valor
  if ( ('a' <= x) && (x <= 'z') )
  {
    IO_printf ( "%s (%c)\n", "Letra minuscula", x );
  }
  else
  {
    IO_printf ( "%s (%c)\n", "Valor diferente de minuscula", x );
    if ( ('A' <= x) && (x <= 'Z') )
    {
      IO_printf ( "%s (%c)\n", "Letra maiuscula", x );
    }
    else
    {
      if ( ('0' <= x) && (x <= '9') )
      {
        IO_printf ( "%s (%c)\n", "Algarismo", x );
      }
      else
      {
        IO_printf ( "%s (%c)\n", "Valor diferente de algarismo", x );
      } // end if
    } // end if
  } // end if

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method_05 ( )
```

```

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

```

- a.) a
- b.) A
- c.) 0
- d.) #

```

----- historico

```

Versao	Data	Modificacao
0.1	__/__/__	esboco

```

----- testes

```

Versao	Teste	
0.1	01. ( OK )	identificacao de programa

```

*/

```

DICA: Melhor usar parênteses para identificar cada condição e dar ênfase à conjunção lógica (**&&=AND**) necessária para se avaliar o pertencimento ao intervalo. Recomenda-se, mais uma vez, o uso da endentação dos blocos a fim de se proporcionar melhor identificação das vinculações entre diversas condições.

- 17.) Compilar o programa novamente.  
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.  
Se não houver erros, seguir para o próximo passo.
- 18.) Executar o programa.  
Observar as saídas.  
Registrar os resultados.
- 19.) Incluir outro método para combinar condições compostas mediante conectivos lógicos.  
Prever novos testes.

```

/*
  Method_06.
*/
void method_06 ( void )
{
  // definir dado
  char x = '_';          // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_06 - Programa - v0.0" );

  // ler do teclado
  x = IO_readchar ( "Entrar com um caractere: " );

```

```

// testar valor
if ( ( 'a' <= x && x <= 'z' ) ||           // minuscula OU
    ( 'A' <= x && x <= 'Z' ) )           // maiuscula
{
    IO_printf ( "%s (%c)\n", "Letra", x );
}
else
{
    IO_printf ( "%s (%c)\n", "Valor diferente de letra", x );
} // end if

// encerrar
IO_pause ( "Apertar ENTER para continuar" );
} // end method_06 ( )

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

a.) 0
b.) 1
c.) 10
d.) -1
e.) 100

----- historico

Versao      Data      Modificacao
0.1         __/___     esboco

----- testes

Versao      Teste
0.1         01. ( OK )   identificacao de programa

*/

```

DICA: Melhor usar parênteses para indicar tudo o que deverá ser negado.

- 20.) Incluir outro método para modificar o teste para usar negação.  
 Observar a inversão dos blocos de comandos.  
 Prever novos testes.

```

/*
  Method_07.
*/
void method_07 ( void )
{
  // definir dado
  char x = '_';          // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_07 - Programa - v0.0" );

  // ler do teclado
  x = IO_readchar ( "Entrar com um caractere: " );

  // testar valor
  if ( ! ( ( 'a' <= x && x <= 'z' ) ||          // NAO (minusculta OU
            ( 'A' <= x && x <= 'Z' ) ) )        //      maiuscula)
  {
    IO_printf ( "%s (%c)\n", "Valor diferente de letra", x );
  }
  else
  {
    IO_printf ( "%s (%c)\n", "Letra", x );
  } // end if

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method_07 ( )

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

a.) 0
b.) 1
c.) 10
d.) -1
e.) 100

----- historico

Versao      Data      Modificacao
0.1         _/_      esboco

----- testes

Versao      Teste
0.1         01. ( OK )      identificacao de programa
*/

```

21.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

22.) Executar o programa.  
Observar as saídas.  
Registrar os valores usados para testes e os resultados.

23.) Incluir outro método para usar alternativa múltipla.  
Prever novos testes.

```
/*
  Method_08.
*/
void method_08 ( void )
{
  // definir dado
  char x = '_';          // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_08 - Programa - v0.0" );

  // ler do teclado
  x = IO_readchar ( "Entrar com um caractere ['0','A','a']: " );

  // testar valor
  switch ( x )
  {
    case '0':
      IO_printf ( "%s (%c=%d)\n", "Valor igual do simbolo zero", x, x );
      break;
    case 'A':
      IO_printf ( "%s (%c=%d)\n", "Valor igual 'a letra A", x, x );
      break;
    case 'a':
      IO_printf ( "%s (%c=%d)\n", "Valor igual 'a letra a", x, x );
      break;
    default: // se nao alguma das opcoes anteriores
      IO_printf ( "%s (%c=%d)\n", "Valor diferente das opcoes ['0','A','a']", x, x );
  } // end switch

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method_08 ( )
```

```

/*
----- documentacao complementar
----- notas / observacoes / comentarios
----- previsao de testes

```

- a.) 0
- b.) A
- c.) a
- d.) 1

```

----- historico

```

Versao	Data	Modificacao
0.1	__/__/__	esboco

```

----- testes

```

Versao	Teste	
0.1	01. ( OK )	identificacao de programa

```

*/

```

- 24.) Compilar o programa novamente.  
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.  
Se não houver erros, seguir para o próximo passo.
- 25.) Executar o programa.  
Observar as saídas.  
Registrar os valores usados para testes e os resultados.
- 26.) Incluir outro método para alterar o tipo de dado usado na alternativa múltipla.  
Prever novos testes.

```

/*
  Method_09.
*/
void method_09 ( void )
{
  // definir dado
  int x = 0;           // definir variavel com valor inicial

  // identificar
  IO_id ( "Method_09 - Programa - v0.0" );

  // ler do teclado
  x = IO_readint ( "Entrar com um inteiro [0,1,2,3]: " );

```

```

// testar valor
switch ( x )
{
    case 0:
        IO_printf ( "%s (%d)\n", "Valor igual a zero", x );
        break;
    case 1:
        IO_printf ( "%s (%d)\n", "Valor igual a um ", x );
        break;
    case 2:
        IO_printf ( "%s (%d)\n", "Valor igual a dois", x );
        break;
    case 3:
        IO_printf ( "%s (%d)\n", "Valor igual a tres", x );
        break;
    default: // se nao for alguma das opcoes anteriores
        IO_printf ( "%s (%d)\n", "Valor diferente das opcoes [0,1,2,3]", x );
} // end switch

// encerrar
IO_pause ( "Apertar ENTER para continuar" );
} // end method_09 ( )

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

a.) 0
b.) 1
c.) 2
d.) 3
e.) 4
f.) -1

----- historico

Versao      Data      Modificacao
0.1         _/_      esboco

----- testes

Versao      Teste
0.1         01. ( OK )      identificacao de programa

*/

```

- 27.) Compilar o programa novamente.  
 Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.  
 Se não houver erros, seguir para o próximo passo.
- 28.) Executar o programa.  
 Observar as saídas.  
 Registrar os valores usados para testes e os resultados.

- 29.) Incluir outro método para usar formas alternativas.  
Prever novos testes.

```
/*
  Method_10.
*/
void method_10 ( void )
{
{
// definir dado
  int x = 0;           // definir variavel com valor inicial

// identificar
  IO_id ( "Method_09 - Programa - v0.0" );

// ler do teclado
  x = IO_readint ( "Entrar com um inteiro [0,1,2,3]: " );

// testar valor
  switch ( x )
  {
    case 0:
      IO_println ( IO_concat ( "Valor igual a zero (",
                               IO_concat ( IO_toString_d ( x ), ")\\n" ) ) );
      break;
    case 1:
      IO_println ( IO_concat ( "Valor igual a um  (",
                               IO_concat ( IO_toString_d ( x ), ")\\n" ) ) );
      break;
    case 2:
      IO_println ( IO_concat ( "Valor igual a dois (",
                               IO_concat ( IO_toString_d ( x ), ")\\n" ) ) );
      break;
    case 3:
      IO_println ( IO_concat ( "Valor igual a três (",
                               IO_concat ( IO_toString_d ( x ), ")\\n" ) ) );
      break;
    default: // se nao for alguma das opcoes anteriores
      IO_println ( IO_concat ( "Valor diferente das opcoes [0,1,2,3] (",
                               IO_concat ( IO_toString_d ( x ), ")" ) ) );
  } // end switch

// encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // end method_10 ( )
```



/\*

----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

a.) 0

b.) 1

c.) 2

d.) 3

e.) 4

f.) -1

----- historico

Versao	Data	Modificacao
0.1	__/__/__	esboco

----- testes

Versao	Teste	
0.1	01. ( OK )	identificacao de programa

\*/

30.) Compilar e testar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, testar o programa, anotar os dados e os resultados e seguir em frente.

Exercícios:

DICAS GERAIS: Consultar o Anexo C 02 na apostila para outros exemplos.

Prever, testar e registrar todos os dados e os resultados obtidos.

Montar todos os métodos em um único programa conforme o último exemplo.

Sugestão: Usar alternativas duplas quando possível.

01.) Incluir um procedimento (0211) para:

- ler um valor inteiro do teclado e
- dizer se é par ou ímpar.

DICA: Considerar o zero como par.

Exemplos: { -6, -3, 0, 3, 6, 9 }

02.) Incluir um procedimento (0212) para:

- ler um valor inteiro do teclado e
- dizer se é ímpar e menor que -15, ou par e maior que 15.

Exemplos: { -60, -13, 0, 13, 26, 39 }

03.) Incluir um procedimento (0213) para:

- ler um valor inteiro do teclado e
- dizer se pertence ao intervalo aberto entre (25:45).

Exemplos: { 15, 25, 30, 35, 65, 70 }

04.) Incluir um procedimento (0214) para:

- ler um valor inteiro do teclado e
- dizer se pertence ao intervalo fechado entre [20:60].

Exemplos: { 5, 15, 20, 45, 60, 65 }

05.) Incluir um procedimento (0215) para:

- ler um valor inteiro do teclado e
- dados os intervalos [10:25] e (15:50),
- dizer se pertence à interseção ou a apenas a um deles.

Exemplos: { 5, 15, 20, 25, 30, 60 }

06.) Incluir um procedimento (0216) para:

- ler dois valores inteiros do teclado e
- dizer se o primeiro é par e o segundo é ímpar.

Exemplos: { (5, 15), (35, 40), (60, 72), (50, 63), (89, 98) }

07.) Incluir um procedimento (0217) para:

- ler dois valores inteiros do teclado e dizer se o primeiro é ímpar e negativo, e se o segundo é par e positivo.

Exemplos: { (-5, -15), (-13, 20), (-30, 45), (60, 72), (-89, -98) }

08.) Incluir um procedimento (0218) para:

- ler dois valores reais do teclado e dizer se o primeiro é menor, igual ou maior que a metade do segundo.

Exemplos: { (0.5, 1.5), (3.0, 3.0), (-5.5, 6.4), (7.8, -8.7) }

09.) Incluir um procedimento (0219) para:

- ler três valores reais do teclado e dizer se o segundo está entre o primeiro e o último, quando esses dois forem diferentes entre si.

OBS.: Notar a ordem dos testes.

Exemplos: { (0.75, 0.5, 1.8), (3.6, 4.5, 2.4), (6.3, 7.2, 6.5), (9.8, 8.9, 8.9) }

10.) Incluir um procedimento (0220) para:

- ler três valores reais do teclado e dizer se o segundo não está entre o primeiro e o último, quando todos forem diferentes entre si.

Exemplos: { (0.75, 0.5, 1.8), (3.6, 4.5, 2.4), (6.3, 7.2, 6.5), (9.8, 8.9, 8.9) }

## Tarefas extras

E1.) Incluir um procedimento (02E1) para:

- ler três valores literais (caracteres) do teclado e dizer se o primeiro valor lido está entre os outros dois, ou se é igual a um deles.

Exemplos: { ('a','e','c'), ('e','a','c'), ('a','c','e'), ('e','c','a'), ('a','e','a') }

E2.) Incluir um procedimento (02E2) para:

- ler três valores literais (caracteres) do teclado e dizer se o primeiro valor lido está fora do intervalo definido pelos outros dois, se esses forem diferentes entre si.
- OBS.: Notar que não haverá garantias de ser o segundo menor que o terceiro.

Exemplos: { ('a','e','c'), ('e','a','c'), ('a','c','e'), ('e','c','a'), ('a','e','a') }