Programação

Flavius Gorgônio
flavius@dct.ufrn.br

Agenda

- Tipos de operadores
  - Aritméticos (unários e binários)
  - Relacionais
  - Lógicos
- Estruturas de controle de fluxo
  - Estruturas de decisão simples
  - Estruturas de decisão compostas
  - Decisões compostas encadeadas
  - Decisão por seleção múltipla

Operadores aritméticos mais comuns no C

Operador	Descrição	Ordem de Execução
%	Operador Módulo	Da esquerda para a direita
/	Divisão	Da esquerda para a direita
*	Multiplicação	Da esquerda para a direita
+	Adição	Da esquerda para a direita
-	Subtração	Da esquerda para a direita

Todos os operadores aritméticos

Operador	Descrição	Operador	Descrição
`a % b`	Módulo	`a %= b`	Atribuição por módulo
`a / b`	Divisão	`a /= b`	Atribuição por divisão
`a * b`	Multiplicação	`a *= b`	Atribuição por multiplicação
`a + b`	Adição	`a += b`	Atribuição por adição
`a - b`	Subtração	`a -= b`	Atribuição por subtração
`++a`	Incremento pré-fixado	`--a`	Decremento pré-fixado
`a++`	Incremento pós-fixado	`a--`	Decremento pós-fixado
`+a`	Adição unária	`-a`	Subtração unária

Operadores relacionais no C

Operador	Descrição	Ordem de Execução
==	Igual a	Da esquerda para a direita
!=	Diferente de	Da esquerda para a direita
>	Maior que	Da esquerda para a direita
>=	Maior ou igual a	Da esquerda para a direita
<	Menor que	Da esquerda para a direita
<=	Menor ou igual a	Da esquerda para a direita

Operadores lógicos mais comuns no C

Operador	Descrição	Ordem de Execução
!	NOT (negação)	Da esquerda para a direita
&&	AND (conjunção)	Da esquerda para a direita
||	OR (disjunção)	Da esquerda para a direita

Todos os operadores lógicos do C

Operador	Descrição	Operador	Descrição
`!a`	NOT lógico	`~a`	Complemento
`a && b`	AND lógico	`a & b`	AND lógico (sobre bits)
`a || b`	OR lógico	`a &= b`	Atribuição AND lógico (sobre bits)
		`a | b`	OR lógico (sobre bits)
		`a |= b`	Atribuição OR lógico (sobre bits)
		`a ^ b`	XOR lógico (sobre bits)
		`a ^= b`	Atribuição XOR lógico (sobre bits)

Tipos de dados de expressões em C

É importante conhecer as regras de tipos para expressões aritméticas e lógicas

Execute o código ao lado, escrito em C, e verifique como funciona as regras de tipos para expressões aritméticas e lógicas nessa linguagem

```c
#include <stdio.h>

int main(void) {
  int a, b;
  float c, d;

  a = 1/2;
  b = 2/2;
  c = 3/2;
  d = 4.0/2;
  printf("a = %d\n", a);
  printf("b = %d\n", b);
  printf("c = %f\n", c);
  printf("d = %f\n", d);
  printf("a == b: %d\n", a == b);
  printf("a < c: %d\n", a < c);
  return 0;
}
```

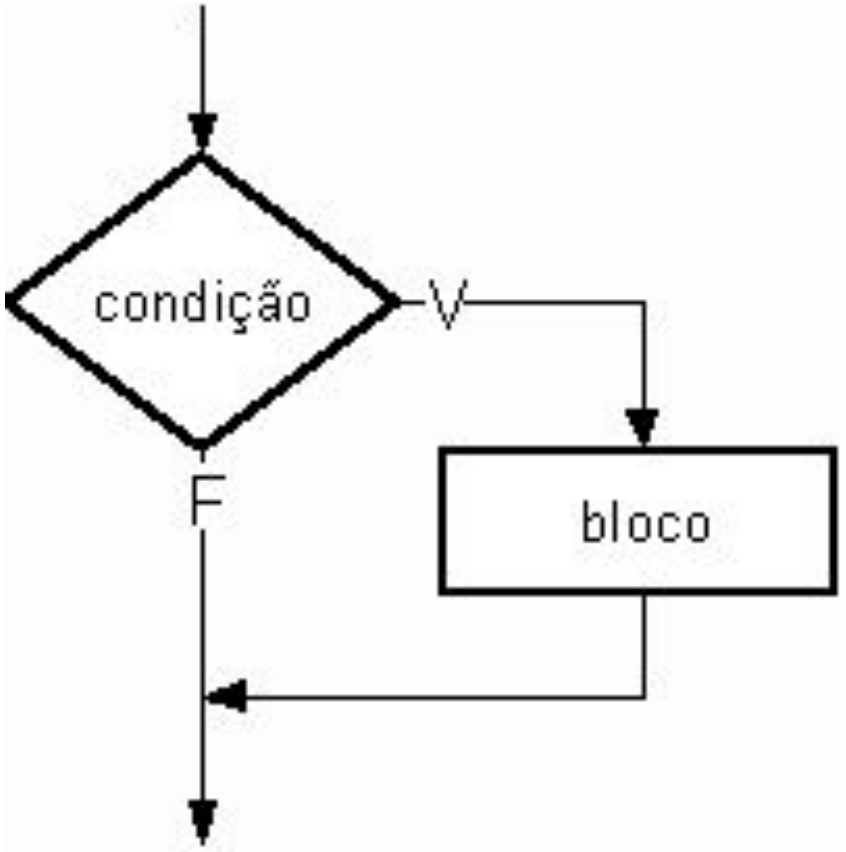
https://repl.it/@flaviusgorgonio/Sem3Ex1c#main.c

Decisão simples

Na sua forma mais simples, permite executar ou não um ou mais comandos (bloco de comandos)

- Se a expressão resultar em VERDADEIRO, o comando (ou bloco de comandos) subordinado à estrutura é executado
- Se resultar em FALSO, o comando (ou bloco de comandos) subordinado à estrutura não é executado

Em ambos os casos, a execução continua com os comandos subsequentes à estrutura de decisão



Comando if na linguagem C

```
...

if (<expressão_booleana>) {

    <bloco_de_comandos>

}

...
```

Expressões booleanas são normalmente construídas a partir de operadores relacionais e/ou lógicos

Na linguagem C, o valor ZERO representa FALSO e qualquer valor diferente de zero representa VERDADEIRO

Assim, em C, operadores aritméticos e expressões podem ser utilizados para a tomada de decisão

Ex:

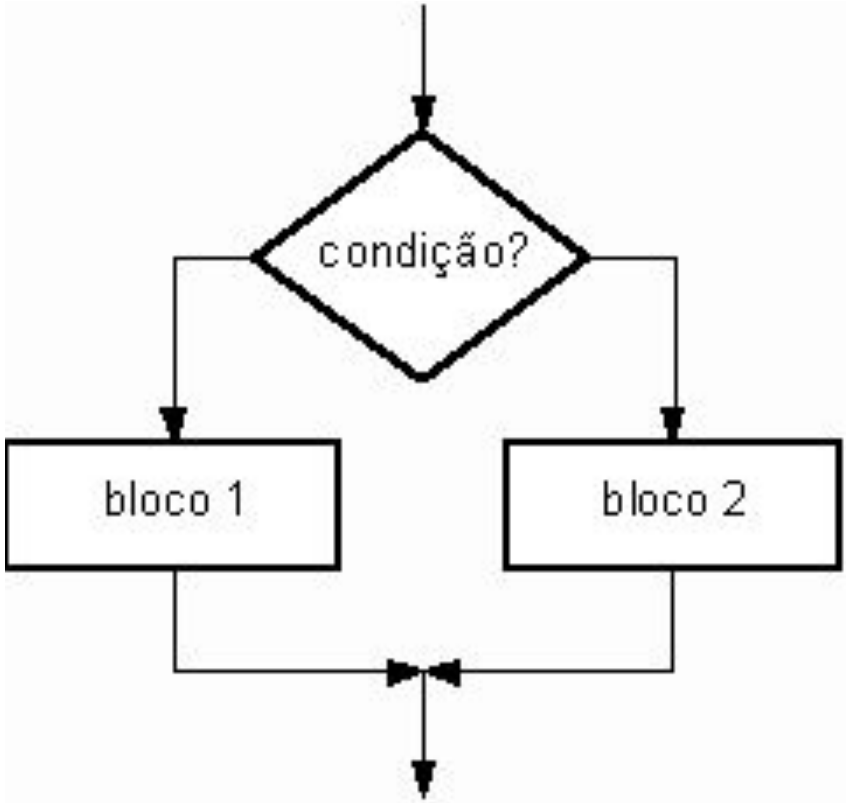
```
if (num_vidas){

    printf("O jogo continua...");

}
```

Decisão composta

Na estrutura de decisão composta, existem dois comandos (ou blocos de comandos). O primeiro segue imediatamente após à expressão e o segundo segue após a cláusula SENÃO (else, na maioria das linguagens)

- Se a expressão resultar em VERDADEIRO, o primeiro comando (ou bloco de comandos) diretamente subordinado à estrutura de decisão é executado
- Se resultar em FALSO, o segundo comando (ou bloco de comandos) diretamente subordinado à cláusula else é executado

Nesse caso, sempre um, e somente um, dos dois comandos (ou blocos de comandos) é executado. Como na estrutura anterior, a execução continua com os comandos subsequentes à estrutura de decisão



Comando if...else na linguagem C

```
...

if (<expressão_booleana>) {

    <bloco_de_comandos_1>

} else {

    <bloco_de_comandos_2>

}

...
```

Como na estrutura anterior, expressões com operadores aritméticos, relacionais e lógicos podem ser utilizados para a tomada de decisão

Ex:

```
if (num_vidas) {

    printf("O jogo continua...");

} else {

    printf("O jogo acabou!");

}
```

Decisões compostas encadeadas

A maioria das LPs permite o encadeamento de estruturas de decisão compostas, onde vários comandos (ou blocos de comandos) são sequenciados para a construção de uma estrutura única de decisão exclusiva dentre múltiplas condições. Nesse caso, após a cláusula SENÃO segue outro teste SE

Se a expressão resultar em VERDADEIRO, o comando (ou bloco de comandos) diretamente subordinado ao teste é executado

Se resultar em FALSO, o próximo teste é verificado e, como no caso anterior, se a expressão resultar em VERDADEIRO, o comando (ou bloco de comandos) diretamente subordinado ao teste é executado

Se resultar em FALSO, passa-se ao teste seguinte e assim por diante

Como na estrutura anterior, se o último bloco for um SENÃO, então apenas UM dentre os comandos (ou blocos de comandos) é executado

Comando if...else if ... na linguagem C

```
...
if (<expressão_booleana_1>) {
    <bloco_de_comandos_1>
}
else if (<expressão_booleana_2>) {
    <bloco_de_comandos_2>
}
...
else {
    ...
}
```

Ex:

```
if (num_vidas >= 2) {
    printf("O jogo continua...");
}
else if (num_vidas == 1) {
    printf("Tenha cuidado!");
}
else {
    printf("O jogo acabou!");
}
```

Sentenças simples e compostas

Em diversas linguagens de programação mais modernas, as cláusulas então e senão podem omitir os delimitadores quando incluem no seu corpo apenas uma sentença simples:

```
if (cont == 0) {
     media = 0;
} else {
     media = soma / cont;
}
```

Pode ser escrito como:

```
if (cont == 0)
     media = 0;
else
     media = soma / cont;
```

Nos casos de mais de uma sentença, o uso de delimitadores é obrigatório.

```
if (cont == 0) {
     media = 0;
     printf("Nenhum valor computado");
} else {
     media = soma / cont;
     printf("Média: %.2f\n", media);
}
```

Entretanto, alguns autores sugerem o uso dos delimitadores em todos os casos, para evitar ambiguidades e/ou erros de codificação e interpretação.

Sentenças simples e compostas

A ausência de delimitadores pode induzir ao erro de interpretação:

```
if (cont == 0)
     if (soma == 0)
          media = 0;
else
     media = soma / cont;
```

Este trecho de código é ambíguo, na verdade ele será interpretado da seguinte forma:

```
if (cont == 0)
     if (soma == 0)
          media = 0;
     else
          media = soma / cont;
```

Para evitar este problema, utiliza-se os delimitadores:

```
if (cont == 0) {
     if (soma == 0) {
          media = 0;
     }
} else {
     media = soma / cont;
}
```

Operador ternário

Linguagens cuja sintaxe é baseada em C possuem um operador ternário, que permite escrever expressões condicionais:

```
expressão_1 ? expressão_2 : expressão_3
```

A expressão_1 é interpretada como sendo uma expressão booleana. Se o resultado for verdadeiro, o valor da expressão_2 é atribuído à expressão inteira. Caso contrário, o valor da expressão_3 que será atribuído à expressão inteira.

Exemplo:

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

Assim, o seguinte código:

```
media = (cont == 0) ? 0 : soma / cont;
```

Equivale ao seguinte trecho:

```
if (cont == 0) {
     media = 0;
} else {
     media = soma / cont;
}
```

Decisão por seleção múltipla

Em algumas LPs, o encadeamento de estruturas de decisão compostas pode ser substituído por uma estrutura de seleção múltipla.

Como na estrutura anterior, os comandos (ou blocos de comandos) são sequenciados para a construção de uma estrutura única de decisão mutuamente exclusiva, escolhida a partir de um casamento de padrões.

Como na estrutura anterior, então apenas UM dentre os comandos (ou blocos de comandos) é executado. Na maioria das LPs existe uma cláusula default que é executada quando nenhuma das cláusulas anteriores é atendida.

Em geral, a variável deve pertencer a tipos de dados simples, como inteiro ou tipo caractere

Comando switch...case na linguagem C

```
...

switch(<variável>) {
    case op1:
        <bloco_de_comandos_1>
        break;

    case op2:
        <bloco_de_comandos_2>
        break;

    ...

    default:
        <bloco_de_comandos_n>
        break;

}
```

```
switch(operador) {
    case '+':
        <bloco_de_soma>
        break;

    case '-':
        <bloco_de_subtração>
        break;

    ...

    default:
        <bloco_operação_inválida>
        break;

}
```

Atividade 1 - Validação de datas

Escreva uma função em linguagem C que receba três valores inteiros correspondentes a uma data (dia, mês e ano) e retorne 1 se a data informada for válida ou 0 se for uma data inválida. Considere a possibilidade do ano ser bissexto e, para isso, escreva também uma função bissexto() que retorna 0 ou 1 indicando se o ano informado como parâmetro é bissexto ou não. O protótipo das duas funções é apresentado a seguir:

```
int data_valida(int, int, int);

int bissexto(int);
```

Função para validar data

```c
int data_valida(int dd, int mm, int aa) {
  int maior_dia;
  if (aa < 0 || mm < 1 || mm > 12 || dd < 1) {
    return 0;
  }
  if (mm == 2) {
    if (bissexto(aa)) {
      maior_dia = 29;
    } else {
      maior_dia = 28;
    }
  } else if (mm == 4 || mm == 6 ||
             mm == 9 || mm == 11) {
               maior_dia = 30;
  } else {
    maior_dia = 31;
  }
  if (dd > maior_dia) {
    return 0;
  }
  return 1;
}
```

```c
int bissexto(int aa) {
  if ((aa % 4 == 0) && (aa % 100 != 0)) {
    return 1;
  } else if (aa % 400 == 0) {
    return 1;
  } else {
    return 0;
  }
}


//
// Esta função pode ser escrita de forma mais
// eficiente? Com menos testes? Como?
//
```

https://repl.it/@flaviusgorgonio/Sem3Ex2c#main.c

Atividade 2 - Mais funções para manipular datas

Escreva uma função em linguagem C que receba três valores inteiros correspondentes a uma data (dia, mês e ano) e retorne o dia do ano correspondente à data informada. Considere a possibilidade do ano ser bissexto. Protótipo da função:

```
int dia_do_ano(int, int, int);
```

Escreva uma função em linguagem C que receba três valores inteiros correspondentes a uma data (dia, mês e ano) e retorne o número da semana do ano que corresponde à data informada. Considere a possibilidade do ano ser bissexto. Protótipo da função:

```
int num_da_semana(int, int, int);
```

Escreva uma função em linguagem C que receba seis valores inteiros correspondentes a duas datas (dia1, mês1, ano1, dia2, mês2, ano2) e retorne o número de dias compreendido entre as datas informadas. Considere a possibilidade de haver anos bissextos entre as duas datas. Protótipo da função:

```
int diferenca_de_datas(int, int,
int, int, int, int);
```

Geração de números pseudo-aleatórios

A função rand() retorna um valor pseudo-aleatório:

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int valor;
  ...
  valor = rand();
  printf("Valor sorteado: %d\n", valor);
  ...
  return 0;
}
```

```
>> ./main
Valor sorteado: 1804289383
```

https://repl.it/@flaviusgorgonio/Sem3Ex3ac#main.c

É possível ajustar o valor gerado pela função rand() dentro de um intervalo específico:

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(void) {
  int valor;
  srand(time(NULL));
  printf("Sorteando um valor entre 1 e 100\n");
  valor = 1 + rand()%100;
  printf("Valor sorteado: %d\n", valor);
  return 0;
}
```

https://repl.it/@flaviusgorgonio/Sem3Ex3bc#main.c

Atividade 3 - Geração de números aleatórios

Escreva um programa em C que simule um dado eletrônico. O programa deve possuir pelo menos duas funções, a primeira para sortear um número inteiro (pseudo-aleatório) entre 1 e 6, e a segunda para exibir na tela uma representação do dado a partir de caracteres alfanuméricos, como se o usuário estivesse jogando um dado real, como no exemplo abaixo:

O valor sorteado foi:

```
##########
# *      #
#   *    #
#     *  #
##########
```

```
int dado(void);
void desenha(int);
```

https://repl.it/@flaviusgorgonio/Sem3Ex3c#main.c

Atividade 4 - Um jogo de dados

Escreva um programa em C que simule uma disputa de dados entre o usuário e o computador. Cada jogador deve lançar dois dados e os pontos dos dados devem ser somados.

O programa deve gerar números pseudo-aleatórios para representar os dados do jogador e do computador, exibindo os valores obtidos e identificando quem ganhou a partida.

Utilize as funções desenvolvidas no programa anterior para apresentar o resultado com uma representação gráfica.

Atividade 5 - Mais um joguinho

Escreva um programa em C que simule o tradicional jogo do Zerinho ou Um. Nesse jogo, deve haver três jogadores, onde o primeiro é um jogador humano e os demais podem ser jogadores humanos ou podem ser simulados pelo computador (pergunte ao usuário quantos jogadores são humanos).

Para jogar, cada jogador deve escolher um dos valores: 0 (ZERO) ou 1 (UM); se for um jogador simulado pelo computador, a escolha deve ser feita através de sorteio.

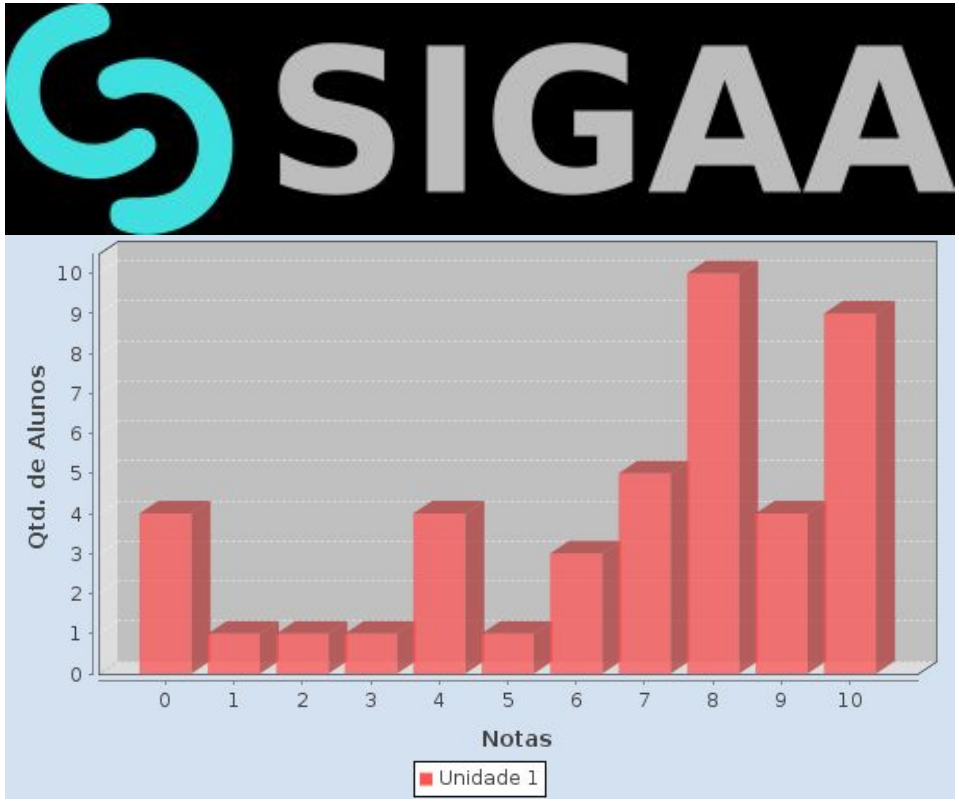
Vence aquele que apresentar um valor distinto de todos os outros. Se todos escolherem números iguais, a partida está empatada.

O programa deve indicar se houve um vencedor ou se a partida acabou empatada.

Atividade 6 - Regras de reposição do SIGAA

Implemente um teste de seleção que verifique a situação de um aluno da UFRN a partir das suas notas em cada unidade, informando se o mesmo está APROVADO, REPROVADO ou se deve realizar a avaliação de REPOSIÇÃO

Consulte o documento “Cartilha do novo regulamento de graduação - UFRN” como referência



Validação de dados em C

A validação de dados de entrada é uma tarefa bastante comum em programação, sendo realizada em praticamente todos os programas que solicitam entrada de dados do usuário.

Na linguagem C, a validação de dados de entrada pode ser feita de várias formas, entre elas:

- Com regras que restringem a entrada de dados pela função **`scanf()`**
- Com leitura e posterior validação manual das entradas
- Com a utilização da biblioteca de expressões regulares <regex.h>

Validação de dados com a função scanf()

```c
#include <stdio.h>

int main(void) {
  char nome[20];
  printf("Informe seu nome: ");
  scanf("%s", nome);
  printf("Bem vindo, %s\n", nome);
  return 0;
}

//
// Exemplo de execução
//

Informe seu nome: Flavius
Bem vindo, Flavius

Informe seu nome: 123deOliveira4
Bem vindo, 123deOliveira4
```

https://repl.it/@flaviusgorgonio/Sem3Ex4ac#main.c

```c
#include <stdio.h>

int main(void) {
  char nome[20];
  printf("Informe seu nome: ");
  scanf("%[A-Za-z]", nome);
  printf("Bem vindo, %s\n", nome);
  return 0;
}

//
// Exemplo de execução
//

Informe seu nome: Flavius123
Bem vindo, Flavius

Informe seu nome: Flavius Gorgonio
Bem vindo, Flavius
```

https://repl.it/@flaviusgorgonio/Sem3Ex4bc#main.c

Validação de dados com a função scanf()

```c
#include <stdio.h>

int main(void) {
  char nome[20];
  printf("Informe seu nome: ");
  scanf("%[A-Z a-z]", nome);
  printf("Bem vindo, %s\n", nome);
  return 0;
}

//
// Exemplo de execução
//

Informe seu nome: Flavius123
Bem vindo, Flavius

Informe seu nome: Flavius Gorgonio
Bem vindo, Flavius Gorgonio
```

https://repl.it/@flaviusgorgonio/Sem3Ex5ac#main.c

```c
#include <stdio.h>

int main(void) {
  char end[20];
  printf("Informe seu endereco: ");
  scanf("%[A-Z a-z.,0-9]", end);
  printf("Endereco: %s\n", end);
  return 0;
}


//
// Exemplo de execução
//

Informe seu endereco: Av. Cel. Martiniano, 123
Endereco: Av. Cel. Martiniano, 123

Informe seu endereco: Av. Cel. Martini@no, 123
Endereco: Av. Cel. Martini
```

https://repl.it/@flaviusgorgonio/Sem3Ex5bc#main.c

Atividade 7 - Validação manual de entradas

```c
#include <stdio.h>

int eh_letra(char);
int eh_digito(char);

int main(void) {
  char letra;
  printf("Informe um caractere: ");
  scanf("%c", &letra);
  printf("Caractere Lido: %c\n", letra);
  printf("Código ASCII: %d\n", letra);
  if (eh_letra(letra)) {
    printf("%c é uma letra\n", letra);
  }
  else if (eh_digito(letra)) {
    printf("%c é um dígito\n", letra);
  }
  else {
    printf("%c é um caractere especial\n", letra);
  }
  return 0;
}
```

```c
int eh_letra(char c) {
  if (c>='A' && c<='Z') {
    return 1;
  }
  else if (c>='a' && c<='z') {
    return 1;
  }
  else {
    return 0;
  }
}

int eh_digito(char c) {
  if (c>='0' && c<='9') {
    return 1;
  }
  else {
    return 0;
  }
}
```

https://repl.it/@flaviusgorgonio/Sem3Ex6c#main.c

Bibliografia Recomendada

Celes, W. *et al*. Introdução a Estruturas de Dados com Técnicas de Programação em C, 2ª ed. Rio de Janeiro: Elsevier, 2016.

Sebesta, R. W., Conceitos de Linguagens de Programação, 9ª ed. Porto Alegre: Bookman, 2016.

Varejão, F., Linguagens de Programação: Conceitos e Técnicas, Rio de Janeiro: Campus, 2004.