Estruturas de dados e estruturas de controle

Estrutura de dados dinâmica

São estruturas que utilizam **alocação dinâmica de memória**, o espaço de memória necessário para armazenar a estrutura do dado não é constante, podendo aumentar ou diminuir sempre que um novo dado é adicionado ou removido da estrutura.

Exemplos de estruturas dinâmicas são:

Filas, pilhas, listas encadeadas, árvores e tabelas de dispersão.

Estruturas de dados e estruturas de controle

As estruturas de controle são utilizadas para manipular e controlar as estruturas de dados, ou seja, controlar o fluxo de informação, e também para controlar as tomadas de decisão dentro de um programa ou algoritmo.

Estrutura de controle sequencial:

```
// comando1;
// comando2;
// comando3;
```

Código fonte /estrutura_sequencial.c[code/cap1/estrutura_sequencial.c]

```
float nota1 = 6;
float nota2 = 8;
float media = (nota1 + nota2)/2;
printf("Media = %f\n", media);
```

Sintaxe geral da estrutura de controle de decisão simples

```
if (condição){
     // bloco de comandos 1;
}// fechamento da chave
```

Código fonte

```
printf("Digite duas notas não negativas de um(a) aluno(a): ");
scanf("%f %f", &nota1, &nota2);

if (nota1 < 0 || nota2 < 0){
  // Bloco de decisão simples
  printf("Notas não podem ser negativas.\n");
  exit(EXIT_FAILURE);
}
```

Ambiente de desenvolvimento

- https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2016/04/como-instalar-o-ubuntu-com-o-virtualbox.html
- https://itsfoss.com/install-linux-in-virtualbox/

Ambiente de desenvolvimento

VS Code


```
$ sudo apt update
$ sudo apt install snapd
```

Either log out and back in again, or restart your system, to ensure snap's paths are updated correctly.

To test your system, install the hello-world snap and make sure it runs correctly:

```
$ sudo snap install hello-world
hello-world 6.3 from Canonical✓ installed
$ hello-world
Hello World!
```

```
sudo snap install --classic code # or code-insiders
```


```
$ sudo apt update
```

```
$ sudo apt install build-essential
```

```
$ sudo apt-get install manpages-dev
```

```
$ gcc --version
```

Curiosidades

- O C apareceu na década de 70 e o seu “pai” foi Dennis Ritchie no AT&T Bell Labs;
- C é uma linguagem de programação genérica que é utilizada para a criação de programas diversos como processadores de texto, programas para a automação industrial, sistemas operacionais, implementação de protocolos de comunicação, etc.

Palavras reservadas

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Estrutura de um Programa em C

```
/*Diretivas de Pré-processamento*/
#include ....
#define ....
```

```
/*Declarações Globais*/
/*Protótipos de Funções*/
```

```
/* Função principal – marca o início da execução do programa*/
int main( ) {
        declarações locais;
        comandos;
         ....
}
```

```
/*Funções*/
Tipo função1 (declaração de parâmetros){
    /*declarações locais;*/
    /*comandos;*/
}
```

Tudo em um arquivo .c

Estruturas de um programa (em C)

Estrutura de um Programa em C

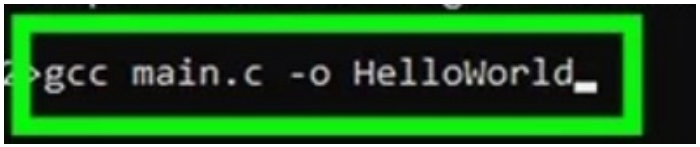
```
/*Diretivas de Pré-processamento*/
#include ....
#define ....

/*Declarações Globais*/
/*Protótipos de Funções*/

/* Função principal – marca o início da execução do programa*/
int main( ) {
        declarações locais;
        comandos;
         ....
}

/*Funções*/
Tipo função1 (declaração de parâmetros){
    /*declarações locais;*/
    /*comandos;*/
}
```

Tudo em um arquivo .c

```c
/*
 * File:  ex.c
 * Author:  mikolas
 * Description: A program exemplifying the use of comments and formatting in C.
*/
#include<stdio.h>

/* The maximum number of values stored. */
#define SZ 100

/* Array of values. */
int vals[SZ];
/* The number of values stored. */
int count;

/* Adds a given value to the global vector returns the new count of values. */
int add(int val) {
    vals[count] = val;
    return ++count;
}

/* Reads n values from stdin and inserts them into the vector vals. */
int main() {
    int n, v;
    scanf("%d", &n);
    while (n--) {
        scanf("%d", &v);
        add(v);
    }
    return 0;
}
```

Primeiro Programa (em C)


```c
#include <stdio.h>

int main(void)
{
    puts("Hello, World");
    return 0;
}
```



Variáveis

- Necessidade de guardar valores na memória;
- Necessidade de aceder a esses valores

Declaração de variáveis

```
tipo_da_variável nome_da_variável;
```

```
int a;
```

Atribuição de valores

```
int a = 5;
```

```
int a, b, c, d;
int e = 5, f = 6;
int g, h = 2, i = 7, j;
```

```
int a;
a = 2;
a = 3;
```

Regras na declaração de variáveis

- Os nomes de variáveis devem ser únicos no mesmo escopo: não podemos ter duas variáveis com o mesmo nome.
- O nome pode ser igual ao de outra variável já existente em escopo superior, porém é recomendado fortemente que não se use variáveis iguais sob pena de tornar o código do programa incompreensível ou de difícil análise;
- Em nomes de variáveis, podemos usar letras maiúsculas ou minúsculas (de A a Z, sem acentos), algarismos arábicos (0-9) e o caractere sublinhado (_), mas o primeiro caractere deve ser uma letra ou o sublinhado.
- Algumas palavras não podem ser usadas para nomes de variáveis por serem palavras reservadas (palavras que têm significado especial na linguagem).
- Não aceita caracteres como $ % @

```
mohd        zara    abc    move_name   a_123
myname50    _temp   j      a23b9       retVal
```

Regras na declaração de variáveis

- O padrão C atual especifica que nomes de até 31 caracteres devem ser aceites. Alguns compiladores podem até aceitar nomes maiores que isso, mas não considere isso uma regra e não use nomes tão longos.
- O C, assim como muitas outras linguagens de programação, é sensível à utilização de maiúsculas e minúsculas(*case sensitive*). Portanto, o código a seguir seria válido e geraria três variáveis diferentes:

```
int nome;
int NOME;
int Nome;
```

Tipos de dados

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	16 byte	3.4E-4932 to 1.1E+4932	19 decimal places

- **Exemplos de como testar**
  - printf(”Dimensão do int: %ld\n", sizeof(int));
  - printf(”Dimensão do int: %ld\n", sizeof(unsigned long));

Operadores Aritméticos

Tabela de Operadores

Operador	Finalidade	Exemplo	Resultado
+	Adição	5 + 2	7
-	Subtração	5 – 2	3
*	Multiplicação	5 * 2	10
/	Divisão (Quociente)	5 / 2	2
%	Divisão Euclidiana (Resto)	30 % 7	2

Precedência

Prioridade	Operador
Alta	! (Operador unário de negação)
Média-alta	* / %
Normal	+ –

a = 8 / 2 * 4

16 ou 1?

16

Forma abreviada

Expressão Original	Expressão equivalente
x=x+k;	x+=k;
x=x-k;	x-=k;
x=x*k;	x*=k;
x=x/k;	x/=k;
x=x>>k;	x>>=k;
x=x<<k;	x<<=k;
x=x&k;	x&=k;;
x=x+1	x+=1 ou x++

Operadores Relacionais e Lógicos

Operadores Relacionais

Operador	Significado
>	Maior que
<	Menor que
>=	Maior ou igual à
<=	Menor ou igual à
==	Igual a
!=	Diferente de

Precedence	Operator
Highest	> >= < <=
Lowest	== !=

Operadores Lógicos

Operator	Name	What It Does
&&	And	Connects two relational expressions. Both expressions must be true for the overall expression to be true.
||	Or	Connects two relational expressions. If either expression is true, the overall expression is true.
!	Not	Reverses the “truth” of an expression, making a true expression false, and a false expression true.

```
if (age <= 12 || age >= 65)
    printf("Admission is free");
else
    printf("You have to pay");
```


```
Operador (da mais alta para a mais baixa)
!
Relacionais (>, >=, <, <=, ==. !=)
&&
||
```

Operadores - Exemplos

```
if (age <= 12 || age >= 65)
    printf("Admission is free");
else
    printf("You have to pay");
```

```
Operador (da mais alta para a mais baixa)
!
Relacionais (>, >=, <, <=, ==. !=)
&&
||
```

- 30 / 7 4
- 7 x 4 + 2 30
- 30 % 7 2
- (7 == 5) falso
- (5 > 4) verdadeiro
- (3 != 2) verdadeiro
- (6 >= 6) verdadeiro

Operações básicas de entrada e saída

Em C, as funções da biblioteca padrão para entrada e saída estão declaradas no cabeçalho **stdio.h**

int puts(const char *str)=> de “put string“ (retorna inteiro não negativo em caso de sucesso)

int putchar(int char)=> de “put char“ (retorna inteiro não negativo em caso de sucesso)

```
puts ("Esta é uma demonstração da função puts.");
putchar ('Z');
```

```
putchar('c');
putchar('h');
putchar('\n');
puts("String.");
puts("Outra string.");
```

int printf(const char *format, ...) => de "print formatted" (imprimir formatado)

```
printf ("string de formatação", arg1, arg2, ...);
```

Suponha que você tem um programa que soma dois valores. Para mostrar o resultado da conta, você poderia fazer isso:

```
int a, b, c;
...          // leitura dos dados
c = a + b;   // '''c''' é o resultado da soma
printf ("%d + %d = %d", a, b, c);
```

O que resultaria em, para a = 5 e b = 9:

```
5 + 9 = 14
```

Valor de Retorno: Inteiro positivo com número de carateres escritos, número negativo em caso de erro

Operações básicas de entrada e saída

- **%d** Número decimal inteiro (int). Também pode ser usado %i como equivalente a %d.
- **%u** Número decimal natural (unsigned int), ou seja, sem sinal.
- **%o** Número inteiro representado na base octal. Exemplo: 41367 (corresponde ao decimal 17143).
- **%x** Número inteiro representado na base hexadecimal. Exemplo: 42f7 (corresponde ao decimal 17143).
- **%X** As letras serão maiúsculas: 42F7.
- **%X** Hexadecimal com letras maiúsculas
- **%f** Número decimal de ponto flutuante. No caso da função printf, devido às conversões implícitas da linguagem C, serve tanto para float como para double. No caso da função scanf, %f serve para float e %lf serve para double.

- **%e** Número em notação científica, por exemplo 5.97e-12. Usar %E para exibir o E maiúsculo (5.97E-12).
- **%E** Número em notação científica com o "e“ maiúsculo.
- **%g** Escolhe automaticamente o mais apropriado entre %f e %e. Usar %G para escolher entre %f e %E.
- **%p** Ponteiro: exibe o endereço de memória do ponteiro em notação hexadecimal.
- **%c** Caractere: imprime o caractere que tem o código ASCII correspondente ao valor dado.
- **%s** Sequência de caracteres (string, em inglês).
- **%%** Imprime um %

- Se o valor for mais largo que o campo, este será expandido para poder conter o valor. O valor nunca será cortado.
- Se o valor for menor que o campo, a largura do campo será preenchida com espaços ou zeros. Os zeros são especificados pela opção 0, que precede a largura.
- O alinhamento padrão é à direita. Para se alinhar um número à esquerda usa-se a opção - (hífen ou sinal de menos) antes da largura do campo.

```
printf ("%*d", num, 300);
```

```
printf ("%-10s", "José");  // exibe "José      "
printf ("%10s", "José");   // exibe "      José"
printf ("%4s", "José");    // exibe "José"
```

```
printf ("%5d", 15);   // exibe "   15"
printf ("%05d", 15);  // exibe "00015"
printf ("%-5d", 15);  // exibe "15   "
```

int scanf(const char *format, ...) => de “scan formatted" (ler formatado)
Retorna número de carateres lidos

```
scanf ("string de formatação", &amp;arg1, &amp;arg2, ...);
```

```
int a;
scanf ("%d", &a);
```

```
int a;
char b;
float c;
scanf ("%d %c %f", &a,&b,&c);
```

```
#include <stdio.h>

int main ()
{
   int a;

   printf ("Digite um número: ");
   scanf ("%d", &a);
   printf ("\nO número digitado foi %d", a);
   return (0);
}
```

Estrutura de controle sequencial:

```
// comando1;
// comando2;
// comando3;
```

Código fonte /estrutura_sequencial.c[code/cap1/estrutura_sequencial.c]

```
float nota1 = 6;
float nota2 = 8;
float media = (nota1 + nota2)/2;
printf("Media = %f\n", media);
```

Exercício 01

Escreva um programa em C que escreva no ecrã "O meu primeiro programa em C" (seguido do carater nova linha "\n")

- Criar um ficheiro ex01.c
- Compilar com o comando `gcc -ansi -Wall -Wextra -pedantic -o ex01.o ex01.c`
- Executar com `./ex01.o`
- O resultado tem que estar de acordo com o ficheiro ./testes/ex01.out
- Executar novamente usando o comando `./ex01.o > ex01.myout`. Verificar que foi criado um novo ficheiro
- Para testar o ficheiro .myout (resultado da execução do programa) com a saída esperada (./testes/ex01.out) utilizar o Comando diff -> `diff ex01.myout ./testes/ex01.out`
- Caso o comando diff não retorne nada, significa que ambos os ficheiros são iguais e que o programa está correto

Exercício 01 Possível Solução

```c

/*
 * File:  ex01.c
 * Author:  Vitor custódio
 * Description: Escrever no ecrã uma mensagem
 */

    #include <stdio.h>

    int main() {
        /*printf() -> escreve no standard display */
        printf("O meu primeiro programa em C\n");
        return 0;
    }
```

Exercício 02

Escreva um programa em C que leia um inteiro e escreva no ecrã "O número é:" (seguido do carater \n)

- Criar um ficheiro ex02.c
- Compilar com o comando `gcc -ansi -Wall -Wextra -pedantic -o ex02.o ex02.c`
- Executar com `./ex02`
- Para a entrada "5", o resultado tem que estar de acordo com o ficheiro ./testes/ex02.out
- Executar novamente usando o comando `./ex02.o < ./testes/ex02.in > ex02.myout`. Verificar que foi criado um novo ficheiro. *Neste caso notar que o ficheiro ex02.in serviu de entrada ao nosso programa, sendo que o output foi redirecionado para o ficheiro .myout*
- Para testar o ficheiro .myout (resultado da execução do programa) com a saída esperada (./testes/ex02.out) utilizar o Comando diff -> `diff ex02.myout ./testes/ex02.out`

Exercício 02 Possível Solução

```c
/*
 * File:  ex02.c
 * Author:  Vitor custódio
 * Description: Escrever no ecrã um numero pedido
 * Last Change: 22Jan2020
 */

   #include <stdio.h>

   int main() {
       /* Variável local que guarda número pedido*/
       int num;
       /* Espera ler um inteiro*/
       scanf("%d",&num);
       /*printf() -> escreve no standard display */
       printf("O número é:%d\n",num);
       return 0;
   }
```

Exercício 03

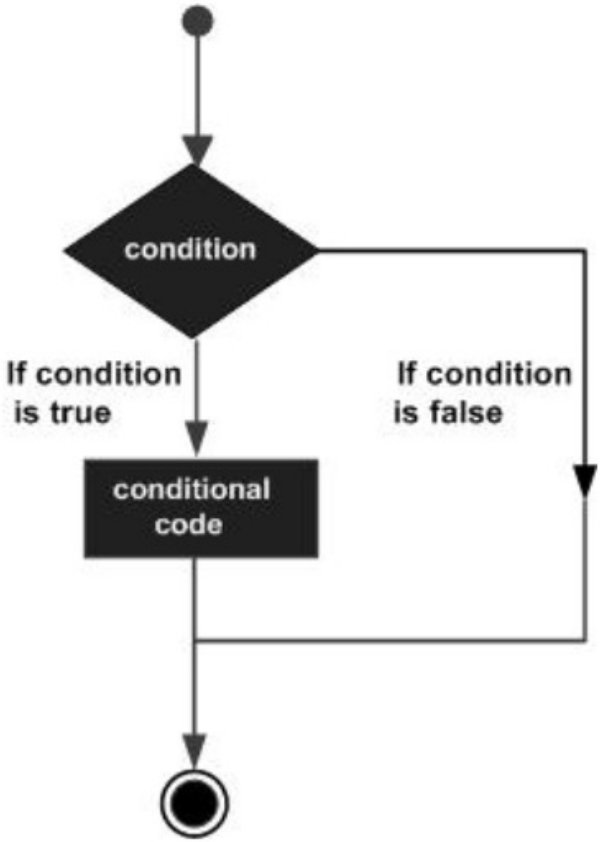
Escreva um programa em C que escreva no ecrã a dimensão (em bytes) de vários tipos de dados, com o seguinte formato: Dimensão char:x
Dimensão int:x
Dimensão float:x
Dimensão double:x
Dimensão long int:x
Dimensão short int:x
Dimensão unsigned long:x
Dimensão long double:x

- Criar um ficheiro ex03.c
- Compilar com o comando `gcc -ansi -Wall -Wextra -pedantic -o ex03.o ex03.c`
- Executar com `./ex03.o`
- O resultado tem que estar de acordo com o ficheiro ./testes/ex03.out
- Executar novamente usando o comando `./ex03.o > ex03.myout`. Verificar que foi criado um novo ficheiro
- Para testar o ficheiro .myout (resultado da execução do programa) com a saída esperada (./testes/ex03.out) utilizar o Comando diff -> `diff ex03.myout ./testes/ex03.out`
- Caso o comando diff não retorne nada, significa que ambos os ficheiros são iguais e que o programa está correto

Exercício 03 Possível Solução

```c
/*
 * File:  ex03.c
 * Author:  Vitor custódio
 * Description: Escrever no ecrã dimensões de tipos de dados
 * Last Change: 22Jan2020
 */

#include <stdio.h>

int main() {
    /*printf() -> escreve no standard display */
    printf("Dimensão char:%ld\n",sizeof(char));
    printf("Dimensão int:%ld\n",sizeof(int));
    printf("Dimensão float:%ld\n",sizeof(float));
    printf("Dimensão double:%ld\n",sizeof(double));
    printf("Dimensão long int:%ld\n",sizeof(long));
    printf("Dimensão short int:%ld\n",sizeof(short));
    printf("Dimensão unsigned long:%ld\n",sizeof(unsigned long));
    printf("Dimensão long double:%ld\n",sizeof(long double));
    return 0;
}
```

Estruturas de Controle – Decisão Simples

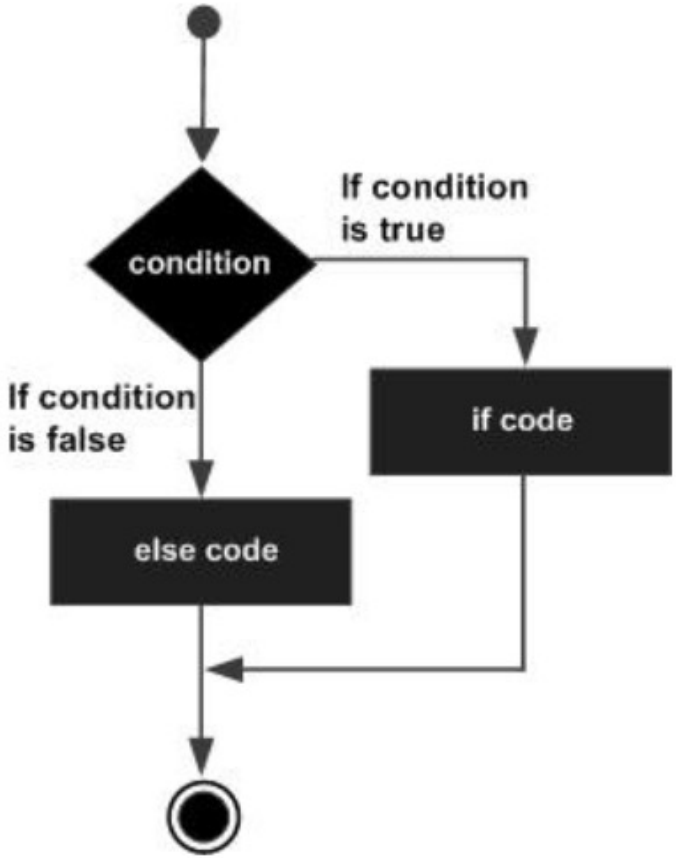


Sintaxe geral da estrutura de controle de decisão simples

```
if (condição){
     // bloco de comandos 1;
}// fechamento da chave
```

Código fonte

```
    printf("Digite duas notas não negativas de um(a) aluno(a): ");
    scanf("%f %f", &nota1, &nota2);

    if (nota1 < 0 || nota2 < 0){
      // Bloco de decisão simples
      printf("Notas não podem ser negativas.\n");
      exit(EXIT_FAILURE);
    }
```


condition
If condition
is true
If condition
is false
if code
else code

Sintaxe geral da estrutura de decisão composta

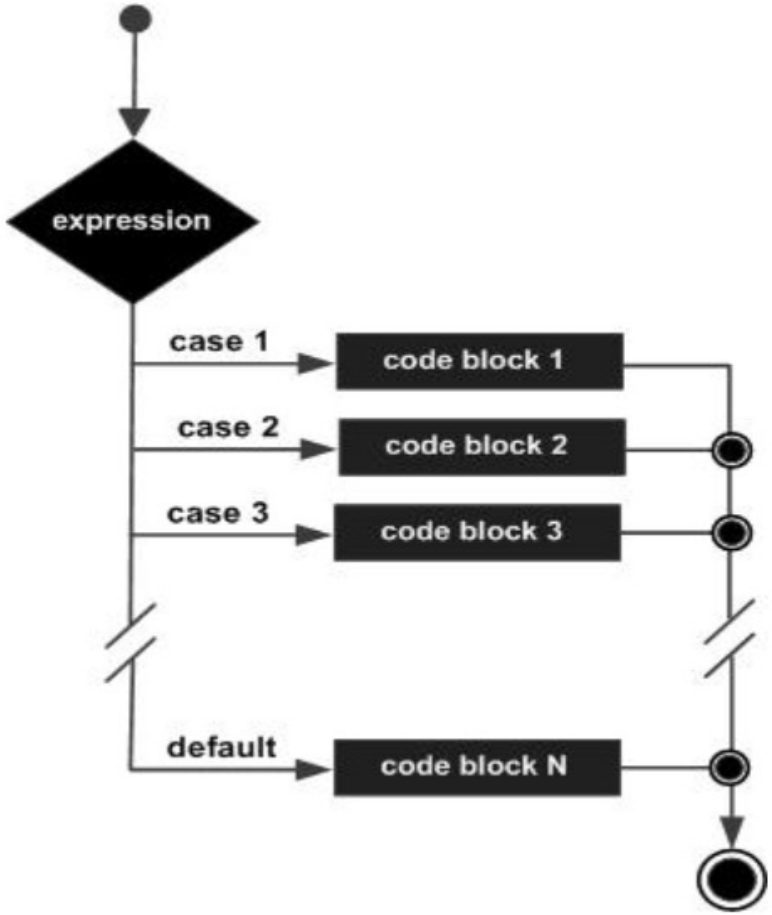
```
// Seleção composta com apenas um else
if (condição1){
    // bloco de comandos 1;
} else {
    // bloco de comandos 2;
}

// seleção composta com else if
if (condição1) {
    // bloco de comandos 1;
} else if (condição2) {
    // bloco de comandos 2;
}  else if (condição3) {
    // bloco de comandos 3;
} else if (condiçãoN) {
    // bloco de comandos N;
} else {
   // bloco de comando default;
}
```

```c
// decisão composta
if (nota1 < 0 || nota1 >10 || nota2 < 0 || nota2 >10){//condição1
  // bloco de comandos 1
  printf("Alguma nota foi inválida: %f, %f\n", nota1, nota2);
  exit(EXIT_FAILURE);
} else if (media>=7.0 && media<=10){//condição2
  // bloco de comandos 2
  printf("Situação do aluno(a): APROVADO(A)\n");
} else if (media>=4 && media<7){//condição3
  // bloco de comandos 3
  printf("Situação do aluno(a): PRECISA FAZER PROVA FINAL\n");
} else {
  // bloco de comandos default
  printf("Situação do aluno(a): REPROVADO(A)\n");
}
```

```c
int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   /* check the boolean condition */
   if( a == 100 ) {

      /* if condition is true then check the following */
      if( b == 200 ) {
         /* if condition is true then print the following */
         printf("Value of a is 100 and b is 200\n" );
      }
   }

   printf("Exact value of a is : %d\n", a );
   printf("Exact value of b is : %d\n", b );

   return 0;
}
```

Sintaxe geral da estrutura de decisão múltipla

```
switch (variável) {
  case VALOR1:
    instruçao1;
    instrução2;
    break;
  case VALOR2:
    instruçao3;
    instrução4;
    break;
  default:
    instruçao5;
    instrução6;
    break;
}
```



Código fonte

```
switch (caracter) {
  case 'a':
  case 'e':
  case 'i':
  case 'o':
  case 'u':
      printf("Você digitou uma vogal minúscula\n");
      break;
  case 'A':
  case 'E':
  case 'I':
  case 'O':
  case 'U':
      printf("Você digitou uma vogal MAIÚSCULA\n");
      break;
  default:
      printf("Você não digitou uma vogal\n");
}
```