ANO
2023

UNINTER

LINGUAGEM DE PROGRAMAÇÃO

LISTA DE EXERCÍCIOS – AULA 1

Prof. Winston Sen Lun Fung, Esp.

LISTA DE EXERCÍCIOS - AULA 1

1. Crie um programa em C que declare variáveis de cada um dos tipos básicos atribua valores as variáveis criadas e, em seguida, imprima seus valores.

2. Crie um programa em C que peça ao usuário que insira um caractere e, em seguida, imprima o código ASCII correspondente ao caractere digitado.

3. Crie um programa em C que peça ao usuário que insira dois números inteiros e imprima a soma, a diferença, o produto e o quociente.

4. Crie um programa em C que peça ao usuário que insira um número inteiro e, em seguida, imprima a tabuada desse número de 1 a 10.

5. Crie um programa em C que peça ao usuário que insira a base e a altura de um triângulo (números de ponto flutuante), e depois calcule e imprima a área desse triângulo.

6. Crie um programa em C que peça ao usuário que insira três números inteiros, calcule a média como um número de ponto flutuante e imprima o resultado.

7. Crie um programa em C que declare uma variável int e uma variável char. Atribua a eles valores fornecidos pelo usuário. Em seguida, incremente cada um várias vezes (o número de vezes também é fornecido pelo usuário) e imprima os resultados.

8. Crie um programa que declare um array (vetor) de 5 inteiros, permita que o usuário preencha esse array, e ao final imprima os números na ordem inversa.

9. Crie um programa que declare um array de 10 inteiros, preencha o array com números de 1 a 10, e então imprima a soma de todos os elementos.

10. Crie um programa que declare um array de 5 números de ponto flutuante, permita que o usuário preencha o array, e então imprima o maior e o menor número.

11. Crie um programa que declare um array de 30 caracteres para receber um nome, permita que o usuário digite o seu nome, e então imprima os caracteres na ordem em que foram inseridos e em ordem inversa.

12. Crie um programa que peça ao usuário para inserir duas strings. Depois, o programa deve concatenar (juntar) essas duas strings e imprimir o resultado.

13. Crie um programa que peça ao usuário para inserir uma string e uma letra. O programa deve então contar quantas vezes essa letra aparece na string.

14. Crie uma função que recebe dois números inteiros e retorna a soma deles. Utilize essa função em seu programa principal.

15. Crie uma função chamada "parOuImpar" que recebe um número inteiro e imprime se ele é par ou ímpar. Use esta função em seu programa principal.

16. Crie uma função que recebe um array de inteiros e seu tamanho e retorna a soma de todos os elementos do array. Use essa função em seu programa principal.

17. Crie um programa que declare uma variável global e uma função. A função deve modificar o valor da variável global para o valor informado pelo usuário e o programa principal deve imprimir este valor.

18. Crie uma função que declare uma variável estática. A função deve incrementar o valor da variável estática e imprimir o seu valor. Chame esta função várias vezes no seu programa principal e observe o comportamento da variável estática.

19. Crie um programa que declare duas variáveis globais. Uma função deve modificar o valor dessas variáveis. Outra função deve imprimir o valor dessas variáveis.

20. Crie uma função que declare uma variável estática e outra função que declare uma variável automática (não-estática). Ambas as funções devem incrementar o valor das respectivas variáveis e imprimir o seu valor. Chame estas funções várias vezes no seu programa principal e observe o comportamento das variáveis estática e automática.

RESPOSTAS DA LISTA DE EXERCÍCIOS

1. Crie um programa em C que declare variáveis de cada um dos tipos básicos atribua valores as variáveis criadas e, em seguida, imprima seus valores.

```c
#include <stdio.h>

int main() {
	// Declaração e inicialização de variáveis
	// Variável inteira i com valor 10
	int i = 10;
	// Variável de ponto flutuante f com valor 12.5
	float f = 12.5;
	// Variável de ponto flutuante com precisão dupla d com valor 20.15
	double d = 20.15;
	// Variável char c com valor 'a'
	char c = 'a';

	// Imprimindo os valores armazenados nas variáveis
	// Exibe o valor inteiro de i
	printf("\nO valor armazenado na variavel i eh : %d", i);
	// Exibe o valor de ponto flutuante de f
	printf("\nO valor armazenado na variavel f eh : %f", f);
	// Exibe o valor de ponto flutuante de precisão dupla de d
	printf("\nO valor armazenado na variavel d eh : %lf", d);
	// Exibe o caractere armazenado em c
	printf("\nO valor armazenado na variavel c eh : %c", c);

	return 0;
}
```

2. Crie um programa em C que peça ao usuário que insira um caractere e, em seguida, imprima o código ASCII correspondente ao caractere digitado.

```c
#include <stdio.h>

int main() {
    char c; // Declaração da variável para armazenar o caractere digitado pelo usuário

    // Solicita ao usuário que digite um caractere
    printf("Digite um caractere: ");
    // Lê o caractere digitado pelo usuário e o armazena na variável 'c'
    scanf_s("%c", &c);

    // Exibe o caractere e o seu código ASCII correspondente
    printf("O código ASCII do caractere %c é %d.\n", c, c);

    return 0;
}
```

Conhecendo a tabela ASCII

A tabela ASCII (American Standard Code for Information Interchange) é uma tabela de referência que associa números (códigos) a caracteres. Esses caracteres podem ser letras, números, símbolos especiais e caracteres de controle, como espaços, tabulações e quebras de linha.

Essa tabela é muito útil porque permite que os computadores representem e entendam caracteres por meio de números. Cada caractere tem um número exclusivo associado a ele na tabela ASCII. Por exemplo, a letra "A" é representada pelo número 65, o símbolo "&" pelo número 38, o número "1" pelo número 49 e assim por diante.

A tabela ASCII é amplamente utilizada em sistemas de computadores e linguagens de programação para armazenar, transmitir e processar texto. Ela fornece uma maneira padronizada de representar caracteres e garante que um mesmo caractere tenha a mesma representação numérica em diferentes sistemas e plataformas.

No exercício, o programa solicita ao usuário que digite um caractere. Em seguida, ele utiliza a tabela ASCII para determinar o código numérico associado a esse caractere e exibe esse código na saída. Por exemplo, se o usuário digitar "A", o programa utilizará a tabela ASCII para encontrar o código correspondente, que é 65, e exibirá essa informação na tela.

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Caracteres de Controle:
Os caracteres de controle na tabela ASCII são os primeiros 32 caracteres, cujos códigos variam de 0 a 31 (inclusive). Esses caracteres não são imprimíveis e são usados principalmente para controlar a formatação e o funcionamento de dispositivos de saída, como impressoras e terminais. Alguns exemplos de caracteres de controle são:
Código 10: Nova linha (LF - Line Feed) - usado para mover o cursor para a próxima linha.
Código 13: Retorno de Carro (CR - Carriage Return) - usado para mover o cursor para o início da linha.
Código 27: Escape (ESC) - usado para iniciar sequências de escape para controle de dispositivos.

Caracteres Imprimíveis:
Os caracteres imprimíveis na tabela ASCII são aqueles que representam letras, números, símbolos e outros caracteres visíveis na tela ou em documentos. Esses caracteres têm códigos que variam de 32 a 126 (inclusive). Alguns exemplos de caracteres imprimíveis são:
Código 65: "A"
Código 97: "a"
Código 48: "0"
Código 33: "!"
Esses são apenas alguns exemplos, e a tabela ASCII contém uma variedade de caracteres imprimíveis.

Caracteres Estendidos:
Os caracteres estendidos na tabela ASCII são aqueles cujos códigos estão acima do valor 127. Esses caracteres foram adicionados posteriormente à tabela ASCII básica para acomodar símbolos e caracteres específicos de outros idiomas ou fins especiais. No entanto, a tabela ASCII padrão contém apenas 128 caracteres, então os caracteres estendidos variam dependendo do conjunto de caracteres e codificação utilizados.
Por exemplo, em conjuntos de caracteres como ISO-8859-1 ou UTF-8, é possível encontrar caracteres acentuados utilizados em línguas como o português (á, é, ã, ç) e caracteres especiais (€, ©, ®) que não estão presentes na tabela ASCII original.

3. Crie um programa em C que peça ao usuário que insira dois números inteiros e imprima a soma, a diferença, o produto e o quociente.

```c
#include <stdio.h>

int main() {
    int num1, num2; // Declaração das variáveis para armazenar os números inteiros

    printf("Informe dois numeros inteiros: "); // Solicita ao usuário que informe dois números inteiros
    printf("\nDigite o primeiro numero: "); // Solicita ao usuário que digite o primeiro número
    scanf_s("%d", &num1); // Lê o primeiro número digitado pelo usuário e o armazena na variável 'num1'
    printf("\nDigite o segundo numero: "); // Solicita ao usuário que digite o segundo número
    scanf_s("%d", &num2); // Lê o segundo número digitado pelo usuário e o armazena na variável 'num2'

    printf("\nSoma: %d", num1 + num2); // Exibe o resultado da soma dos números digitados
    printf("\nDiferenca: %d", num1 - num2); // Exibe o resultado da diferença entre os números digitados
```

```
14.     printf("\nProduto: %d", num1 * num2); // Exibe o resultado do produto dos números digitados
15.     printf("\nQuociente: %d", num1 / num2); // Exibe o resultado do quociente da divisão dos números
digitados
16.
17.     return 0;
18. }
19.
```

4. Crie um programa em C que peça ao usuário que insira um número inteiro e, em seguida, imprima a tabuada desse número de 1 a 10.

```
1. #include <stdio.h>
2.
3. int main() {
4.       // Declaração das variáveis para armazenar o número inteiro e o contador do loop
5.       int num, i;
6.
7.       // Solicita ao usuário que digite um número inteiro
8.       printf("Digite um numero inteiro: ");
9.
10.      // Lê o número digitado pelo usuário e o armazena na variável 'num'
11.      scanf_s("%d", &num);
12.
13.      // Loop 'for' para calcular a tabuada do número digitado
14.      for (i = 1; i <= 10; i++) {
15.               // Exibe a multiplicação do número digitado pelo contador do loop
16.               printf("%d * %d = %d\n", num, i, num * i);
17.      }
18.
19.      return 0;
20. }
```

5. Crie um programa em C que peça ao usuário que insira a base e a altura de um triângulo (números de ponto flutuante), e depois calcule e imprima a área desse triângulo.

```
1. #include <stdio.h>
2.
3. int main() {
4.     // Declaração das variáveis para armazenar a base e a altura do triângulo
5.     float base, altura;
6.
7.     // Solicita ao usuário que digite a base e a altura do triângulo
8.     printf("\nDigite a base e a altura do triangulo: ");
9.
10.    // Solicita ao usuário que digite o valor da base do triângulo
11.    printf("\nDigite o valor para a base do triangulo: ");
12.
13.    // Lê o valor da base digitado pelo usuário e o armazena na variável 'base'
14.    scanf_s("%f", &base);
15.
16.    // Solicita ao usuário que digite o valor da altura do triângulo
17.    printf("\nDigite o valor para a altura do triangulo: ");
18.
19.    // Lê o valor da altura digitado pelo usuário e o armazena na variável 'altura'
20.    scanf_s("%f", &altura);
21.
22.    // Cálculo e exibição da área do triângulo
23.    // Exibe o resultado da área do triângulo
24.    printf("\nA area do triangulo eh: %.2f\n", 0.5 * base * altura);
25.
26.    return 0;
27. }
```

6. Crie um programa em C que peça ao usuário que insira três números inteiros, calcule a média como um número de ponto flutuante e imprima o resultado.

```c
#include <stdio.h>

int main() {
	// Declaração das variáveis para armazenar os três números inteiros
	int num1, num2, num3;
	// Variável para armazenar a média dos números
	float media;

	// Solicita ao usuário que digite três números inteiros
	printf("Digite tres numeros inteiros: ");

	// Solicita ao usuário que informe o primeiro número
	printf("\nInforme o primeiro numero: ");

	// Lê o primeiro número digitado pelo usuário e o armazena na variável 'num1'
	scanf_s("%d", &num1);

	// Solicita ao usuário que informe o segundo número
	printf("\nInforme o primeiro numero: ");

	// Lê o segundo número digitado pelo usuário e o armazena na variável 'num2'
	scanf_s("%d", &num2);

	// Solicita ao usuário que informe o terceiro número
	printf("\nInforme o primeiro numero: ");

	// Lê o terceiro número digitado pelo usuário e o armazena na variável 'num3'
	scanf_s("%d", &num3);

	// Cálculo da média dos números informados
	// Realiza o cálculo da média, convertendo a soma para float para garantir a precisão
	media = (float)(num1 + num2 + num3) / 3;

	// Exibe o resultado da média com duas casas decimais após a vírgula
	printf("A media eh: %.2f\n", media);

	return 0;
}
```

7. Crie um programa em C que declare uma variável int e uma variável char. Atribua a eles valores fornecidos pelo usuário. Em seguida, incremente cada um várias vezes (o número de vezes também é fornecido pelo usuário) e imprima os resultados.

```c
#include <stdio.h>

// Função para limpar o buffer de entrada do teclado
void clearBuffer() {
    int k;
    while ((k = getchar()) != '\n' && k != EOF);
}

int main() {
    // Declaração das variáveis para armazenar o número inteiro
    // o caractere e a quantidade de repetições
    int i, n;
    char c;

    // Solicita ao usuário que digite um número inteiro e um caractere
    printf("\nDigite um número inteiro e um caractere: ");
    // Solicita ao usuário que informe o número inteiro
    printf("\nDigite o número inteiro: ");
    // Lê o número inteiro digitado pelo usuário e o armazena na variável 'i'
    scanf_s("%d", &i);
    // Limpa o buffer de entrada após a leitura do número inteiro
    clearBuffer();

```

```c
    // Solicita ao usuário que informe o caractere
    printf("\nDigite o caractere: ");
    // Lê o caractere digitado pelo usuário e o armazena na variável 'c'
    scanf_s("%c", &c);
    // Limpa o buffer de entrada após a leitura do caractere
    clearBuffer();

    // Solicita ao usuário que informe quantas vezes deseja incrementar o número e o caractere
    printf("\nQuantas vezes voce deseja incrementar o numero e o caractere? ");
    // Lê o número digitado pelo usuário e o armazena na variável 'n'
    scanf_s("%d", &n);
    // Limpa o buffer de entrada após a leitura do número
    clearBuffer();

    // Loop para incrementar o número inteiro 'i' e o caractere 'c' 'n' vezes
    for (int j = 0; j < n; j++) {
        // Incrementa o valor do número inteiro 'i'
        i++;
        // Incrementa o valor do caractere 'c' (o próximo caractere na tabela ASCII)
        c++;
    }

    // Exibe o resultado após os incrementos
    printf("\nApos incrementar %d vezes, o valor inteiro eh %d e o caractere eh %c.\n", n, i, c);

    return 0;
}
```

Problema de Buffer de Teclado:

Quando você usa a função scanf() ou scanf_s() para ler entradas do teclado, como números inteiros ou caracteres, o programa armazena os caracteres digitados pelo usuário em um buffer de entrada. No entanto, a função scanf() ou scanf_s() pode deixar caracteres extras, como o caractere de nova linha (\n), no buffer após a leitura de um valor.

Por exemplo, suponha que você use scanf("%d", &num); para ler um número inteiro. Se você digitar o número e pressionar "Enter", o número será lido corretamente, mas o caractere de nova linha (\n) deixado no buffer de entrada pode causar problemas ao ler entradas subsequentes, como caracteres ou outras strings.

Solução com a Função clearBuffer():

Para evitar esse problema, é recomendável limpar o buffer de entrada do teclado após cada leitura de valor usando scanf() ou scanf_s(). A função clearBuffer() é criada com esse propósito. Ela lê e descarta todos os caracteres armazenados no buffer até encontrar uma nova linha (\n) ou o final do arquivo (EOF), garantindo que o buffer esteja vazio antes de ler novos valores.

Aqui está a definição da função clearBuffer() novamente para referência:

```c
void clearBuffer() {
    int k;
    while ((k = getchar()) != '\n' && k != EOF);
}
```

Essa função é simples, usando um loop while para ler e descartar os caracteres até que a nova linha (\n) ou o final do arquivo (EOF) seja encontrado.

Ao chamar clearBuffer() após a leitura de valores usando scanf_s() como mostrado no exemplo anterior, você garante que o buffer de entrada esteja sempre limpo e pronto para novas leituras, evitando problemas de leitura indesejada e comportamentos inesperados no programa. Isso torna a leitura de entradas do teclado mais robusta e confiável.

8. Crie um programa que declare um array (vetor) de 5 inteiros, permita que o usuário preencha esse array, e ao final imprima os números na ordem inversa.

```c
#include <stdio.h>

int main() {
    // Declaração do array 'numeros' com espaço para armazenar 5 números inteiros
    int numeros[5];
    // Variável de controle do loop 'for'
    int i;

    // Solicita ao usuário que digite 5 números inteiros
```

```
    printf("\nDigite 5 numeros inteiros:");

    for (i = 0; i < 5; i++) {
        // Solicita ao usuário que informe o número atual do loop
        printf("\nDigite o %d numero: ", i + 1);
        // Lê o número inteiro digitado pelo usuário e o armazena no array 'numeros'
        scanf_s("%d", &numeros[i]);
    }

    // Loop para percorrer o array 'numeros' de trás para frente (ordem inversa)
    printf("\nOs numeros em ordem inversa sao: ");
    // Loop para percorrer o array 'numeros' de trás para frente (ordem inversa)
    for (i = 5 - 1; i >= 0; i--) {
        // Exibe o número atual do array 'numeros' no loop
        printf("%d ", numeros[i]);
    }

    return 0;
}
```

9. Crie um programa que declare um array de 10 inteiros, preencha o array com números de 1 a 10, e então imprima a soma de todos os elementos.

```
#include <stdio.h>

int main() {
      // Declaração do array 'numeros' com espaço para armazenar 10 números inteiros
      int numeros[10];
      // Variável para armazenar a soma dos elementos do array 'numeros', inicializada com valor zero
      int soma = 0;

      // Loop 'for' para preencher o array e calcular a soma dos elementos
      for (int i = 0; i < 10; i++) {
                // Preenche o elemento do array na posição 'i' com o valor de 'i + 1'
                numeros[i] = i + 1;

                // O operador '+=' é uma forma abreviada de realizar uma adição e atribuir o
                //  resultado à variável 'soma'.
                // É o mesmo que escrever 'soma = soma + numeros[i];', mas de forma mais concisa.

                // Soma o valor do elemento atual do array 'numeros' à variável 'soma'
                soma += numeros[i];
      }

      // Exibe a soma total dos elementos do array 'numeros'
      printf("\nA soma de todos os elementos eh %d. \n", soma);

      return 0;
}
```

10. Crie um programa que declare um array de 5 números de ponto flutuante, permita que o usuário preencha o array, e então imprima o maior e o menor número.

```
#include <stdio.h>

int main() {
    // Declaração do array 'numeros' com espaço para armazenar 5 números de ponto flutuante (float)
    float numeros[5];
    // Variáveis para armazenar o maior e o menor número do array 'numeros'
    float max, min;
    // Variável de controle do loop 'for'
    int i;

    // Solicita ao usuário que digite 5 números
    printf("\nDigite 5 numeros:");
    for (i = 0; i < 5; i++) {
        // Solicita ao usuário que informe o número atual do loop
        printf("\nInforme o %d numero: ", i + 1);
        // Lê o número de ponto flutuante digitado pelo usuário e o armazena no array 'numeros'
```

```
        scanf_s("%f", &numeros[i]);

        // O bloco 'if-else' abaixo é utilizado para determinar o maior e o menor número enquanto
        // o usuário insere os valores. Se for o primeiro número (i == 0), ele é atribuído tanto
        // a 'max' quanto a 'min' por ser o único valor inserido até o momento.
        // Caso contrário, comparamos o valor atual de 'numeros[i]' com o 'max' e o 'min',
        // atualizando-os se necessário.
        if (i == 0) {
            // Define o primeiro número como o maior e o menor valor atual
            max = min = numeros[i];
        }
        else {
            if (numeros[i] > max) {
                // Atualiza o valor de 'max' se encontrarmos um número maior
                max = numeros[i];
            }
            if (numeros[i] < min) {
                // Atualiza o valor de 'min' se encontrarmos um número menor
                min = numeros[i];
            }
        }
    }

    // Exibe o maior número inserido pelo usuário com duas casas decimais
    printf("\nMaior numero: %.2f", max);
    // Exibe o menor número inserido pelo usuário com duas casas decimais
    printf("\nMenor numero: %.2f", min);

    return 0;
}
```

11. Crie um programa que declare um array de 30 caracteres para receber um nome, permita que o usuário digite o seu nome, e então imprima os caracteres na ordem em que foram inseridos e em ordem inversa.

```
#include <stdio.h>
// A biblioteca string.h é incluída para usar a função strlen().
#include <string.h>

int main() {
    // Declaração do array 'nome' para armazenar o nome digitado pelo usuário
    char nome[30];
    // Variável para armazenar o tamanho do nome (quantidade de caracteres)
    int tam;
    // Variável de controle do loop 'for'
    int i;

    // Solicita ao usuário que digite seu nome
    printf("\nDigite o seu nome: ");
    // Lê o nome digitado pelo usuário e o armazena no array 'nome',
    // limitado a 29 caracteres, reservando a última posição para o terminador de string '\0'
    gets_s(nome, 29);

    // Calcula o tamanho do nome (quantidade de caracteres) usando a função 'strlen'
    tam = strlen(nome);

    // Exibe o cabeçalho para mostrar o nome na ordem inserida (normal)
    printf("\nNome na ordem inserida:\n");
    for (i = 0; i < tam; i++) {
        // Imprime cada caractere do nome na ordem inserida, um por um
        printf("%c", nome[i]);
    }

    // Exibe o cabeçalho para mostrar o nome na ordem inversa
    printf("\n\nNome na ordem inversa:\n");
    for (i = tam - 1; i >= 0; i--) {
        // Imprime cada caractere do nome na ordem inversa, um por um
        printf("%c", nome[i]);
    }

```

```
36.     return 0;
37. }
```

12. Crie um programa que peça ao usuário para inserir duas strings. Depois, o programa deve concatenar (juntar) essas duas strings e imprimir o resultado.

```
1. #include <stdio.h>
2. // A biblioteca string.h é incluída para usar a função strcat_s().
3. #include <string.h>
4.
5. int main() {
6.     // Declaração de duas strings 'str1' e 'str2' com espaço para armazenar até 49 caracteres cada,
7.     // reservando a última posição para o terminador de string '\0'
8.     char str1[50], str2[50];
9.
10.    // Solicita ao usuário que digite a primeira string
11.    printf("\nDigite a primeira string: ");
12.    // Lê a primeira string digitada pelo usuário e a armazena em 'str1', limitada a 49 caracteres
13.    gets_s(str1, 49);
14.
15.    // Solicita ao usuário que digite a segunda string
16.    printf("\nDigite a segunda string: ");
17.    // Lê a segunda string digitada pelo usuário e a armazena em 'str2', limitada a 49 caracteres
18.    gets_s(str2, 49);
19.
20.    // Concatena a segunda string 'str2' à primeira string 'str1'
21.    strcat_s(str1, str2);
22.
23.    // Exibe o resultado da concatenação das strings 'str1'
24.    printf("\nResultado da concatenacao: %s", str1);
25.
26.    return 0;
27. }
```

13. Crie um programa que peça ao usuário para inserir uma string e uma letra. O programa deve então contar quantas vezes essa letra aparece na string.

```
1. #include <stdio.h>
2.
3. int main() {
4.     // Declaração de uma string 'str' com espaço para armazenar até 99 caracteres e uma variável
5.     //'ch' para armazenar a letra digitada pelo usuário
6.     char str[100], ch;
7.     // Variável para contar o número de ocorrências da letra 'ch' na string 'str'
8.     int contador = 0;
9.
10.    // Solicita ao usuário que digite uma string
11.    printf("\nDigite uma string: ");
12.    // Lê a string digitada pelo usuário e a armazena no array 'str', limitada a 99 caracteres
13.    // A última posição do array é reservada para o terminador de string '\0'
14.    gets_s(str, 99);
15.
16.    // Solicita ao usuário que digite uma letra
17.    printf("\nDigite uma letra: ");
18.    // Lê a letra digitada pelo usuário e a armazena na variável 'ch'
19.    scanf_s(" %c", &ch);
20.    // O espaço antes do %c é utilizado para descartar o caractere de nova linha ('\n') ou
21.    //outros espaços em branco que possam ter ficado no buffer após a leitura da string 'str'
22.
23.    // O loop 'for' percorre cada caractere da string 'str' até encontrar o terminador de string '\0'
24.    for (int i = 0; str[i] != '\0'; i++) {
25.        // Compara cada caractere da string 'str' com a letra 'ch' para contar suas ocorrências
26.        if (str[i] == ch) {
27.            // Incrementa o contador se o caractere atual da string for igual à letra 'ch'
28.            contador++;
29.        }
30.    }
31.
32.    // Exibe o resultado, mostrando a letra 'ch' e o número de vezes que ela aparece na string 'str'
```

```
33.     printf("\nA letra '%c' aparece %d vezes na string.\n", ch, contador);
34.
35.     return 0;
36. }
```

14. Crie uma função que recebe dois números inteiros e retorna a soma deles. Utilize essa função em seu programa principal.

```
 1. #include <stdio.h>
 2.
 3. // Declaração da função de soma que recebe dois inteiros e retorna a soma deles
 4. int somar(int num1, int num2) {
 5.     int resultado = num1 + num2;
 6.     // Retorna para a função main() a o resultado da operação num1 + num2
 7.     return resultado;
 8. }
 9.
10. int main() {
11.     int num1, num2;
12.
13.     printf("Digite o primeiro numero: ");
14.     // Lê o primeiro número digitado pelo usuário
15.     scanf_s("%d", &num1); // Lê o primeiro número digitado pelo usuário
16.
17.     printf("Digite o segundo numero: ");
18.     // Lê o segundo número digitado pelo usuário
19.     scanf("%d", &num2);
20.
21.     // Chamada da função somar passando num1 e num2 como argumentos
22.     int resultado = somar(num1, num2);
23.
24.     // Exibe o resultado da soma
25.     printf("A soma dos numeros eh: %d\n", resultado);
26.
27.     return 0;
28. }
```

15. Crie uma função chamada "parOuImpar" que recebe um número inteiro e imprime se ele é par ou ímpar. Use esta função em seu programa principal.

```
 1. #include <stdio.h>
 2.
 3. // Definição da função 'parOuImpar' que recebe um inteiro 'n' como parâmetro
 4. void parOuImpar(int n) {
 5.     // Verifica se 'n' é par ou ímpar e imprime a mensagem correspondente
 6.     if (n % 2 == 0) {
 7.         printf("%d eh par.\n", n);
 8.     }
 9.     else {
10.         printf("%d eh ímpar.\n", n);
11.     }
12. }
13.
14. int main() {
15.     // Declaração de uma variável 'num' do tipo int
16.     int num;
17.
18.     printf("Digite um numero inteiro: ");
19.     // Lê um número inteiro digitado pelo usuário e o armazena em 'num'
20.     scanf_s("%d", &num);
21.
22.     // Chamada da função 'parOuImpar' passando 'num' como argumento
23.     parOuImpar(num);
24.
25.     return 0;
26. }
```

16. Crie uma função que recebe um array de inteiros e seu tamanho e retorna a soma de todos os elementos do array. Use essa função em seu programa principal.

```c
#include <stdio.h>

// Definição da função 'somaArray' que recebe um ponteiro para inteiro 'array'
// e um inteiro 'tamanho' como parâmetros
int somaArray(int* array, int tamanho) {
    int soma = 0;
    for (int i = 0; i < tamanho; i++) {
        // Adiciona o valor do elemento atual do array à variável 'soma'
        soma += array[i];
    }
    // Retorna o valor total da soma dos elementos do array
    return soma;
}

int main() {
    // Declaração e inicialização de um array de inteiros 'numeros' com 5 elementos
    int numeros[5] = { 1, 2, 3, 4, 5 };

    // Chama a função 'somaArray' passando o array 'numeros' e o tamanho 5 como argumentos
    printf("A soma dos elementos do array é %d.\n", somaArray(numeros, 5));

    return 0;
}
```

17. Crie um programa que declare uma variável global e uma função. A função deve modificar o valor da variável global para o valor informado pelo usuário e o programa principal deve imprimir este valor.

```c
#include <stdio.h>

// Declaração de uma variável global
int variavelGlobal;

// Definição da função que modifica o valor da variável global
void modificarVariavelGlobal(int novoValor) {
    // Atribui o novo valor à variável global
    variavelGlobal = novoValor;
}

int main() {
    // Declaração de uma variável local para armazenar o valor informado pelo usuário
    int novoValor;

    printf("Digite um valor inteiro: ");
    // Lê o valor informado pelo usuário e o armazena na variável 'novoValor'
    scanf_s("%d", &novoValor);

    // Chama a função para modificar o valor da variável global
    modificarVariavelGlobal(novoValor);

    // Imprime o valor da variável global após a modificação
    printf("Valor da variavel global: %d\n", variavelGlobal);

    return 0;
}
```

Explicação do código:

1. A biblioteca **stdio.h** é incluída para permitir o uso de funções de entrada/saída padrão em C.
2. É declarada uma variável global **variavelGlobal**, que terá escopo em todo o programa.
3. A função **modificarVariavelGlobal** é definida abaixo da função **main**. Essa função recebe um inteiro **novoValor** como parâmetro e não retorna nenhum valor (por isso é do tipo **void**).

4. Dentro da função **modificarVariavelGlobal**, o novo valor passado como parâmetro é atribuído à variável global **variavelGlobal**.
5. A função **main** é o ponto de entrada do programa.
6. É declarada uma variável local **novoValor** para armazenar o valor informado pelo usuário.
7. O programa exibe uma mensagem pedindo ao usuário que digite um valor inteiro.
8. A função **scanf_s()** é usada para ler o valor inteiro informado pelo usuário e armazená-lo na variável **novoValor**.
9. A função **modificarVariavelGlobal** é chamada passando o valor de **novoValor** como argumento, o que modificará o valor da variável global **variavelGlobal**.
10. O programa imprime o valor atual da variável global após a modificação usando **printf().**

18. Crie uma função que declare uma variável estática. A função deve incrementar o valor da variável estática e imprimir o seu valor. Chame esta função várias vezes no seu programa principal e observe o comportamento da variável estática.

```c
#include <stdio.h>

// Definição da função 'incrementarStatic'
void incrementarStatic() {
    // Variável estática 'num' que mantém seu valor entre chamadas
    static int num = 0;
    // Incrementa o valor da variável estática 'num'
    num++;
    // Imprime o valor atual da variável estática
    printf("Valor atual da variável estática: %d\n", num);
}

int main() {
    // Chama a função 'incrementarStatic'
    incrementarStatic();
    // Chama novamente a função 'incrementarStatic'
    incrementarStatic();
    // Chama mais uma vez a função 'incrementarStatic'
    incrementarStatic();

    return 0;
}
```

Explicação do código:

1. A biblioteca **stdio.h** é incluída para permitir o uso de funções de entrada/saída padrão em C.
2. A função **incrementarStatic** é definida abaixo da função **main()**. Essa função não recebe nenhum parâmetro e não retorna nenhum valor (por isso é do tipo **void**).
3. Dentro da função **incrementarStatic()**, é declarada uma variável estática **num**. A variável estática mantém seu valor entre as chamadas da função, ou seja, ela não é reinicializada a cada vez que a função é chamada.
4. Em cada chamada da função, o valor da variável estática **num** é incrementado em 1.
5. Em cada chamada da função, é impresso o valor atual da variável estática **num** usando **printf()**.
6. A função **main()** é o ponto de entrada do programa.
7. A função **incrementarStatic()** é chamada três vezes em sequência dentro da função **main()**.

19. Crie um programa que declare duas variáveis globais. Uma função deve modificar o valor dessas variáveis. Outra função deve imprimir o valor dessas variáveis.

```c
#include <stdio.h>

// Declaração das variáveis globais
int var1 = 0;
int var2 = 0;

// Função para modificar o valor das variáveis globais
void modificarVariaveisGlobais(int novoValor1, int novoValor2) {
    var1 = novoValor1;
    var2 = novoValor2;
}

// Função para imprimir o valor das variáveis globais
void imprimirVariaveisGlobais() {
    printf("Valor da primeira variavel global: %d\n", var1);
    printf("Valor da segunda variavel global: %d\n", var2);
```

```c
}

int main() {
    // Imprime o valor inicial das variáveis globais através da função 'imprimirVariaveisGlobais'
    imprimirVariaveisGlobais();

    // Modifica o valor das variáveis globais através da função 'modificarVariaveisGlobais'
    modificarVariaveisGlobais(10, 20);

    // Imprime o valor alterado das variáveis globais através da função 'imprimirVariaveisGlobais'
    imprimirVariaveisGlobais();

    return 0;
}
```

Explicação do código:

1. A biblioteca **stdio.h** é incluída para permitir o uso de funções de entrada/saída padrão em C.
2. Duas variáveis globais **var1** e **var2** são declaradas no início do programa.
3. A função **modificarVariaveisGlobais()** é definida abaixo da função **main()**. Essa função recebe dois inteiros **novoValor1** e **novoValor2** como parâmetros e não retorna nenhum valor (por isso é do tipo **void**).
4. Dentro da função **modificarVariaveisGlobais()**, os valores passados como parâmetros são atribuídos às variáveis globais **var1** e **var2**, modificando seus valores.
5. A função **imprimirVariaveisGlobais()** é definida abaixo da função **main()**. Essa função não recebe nenhum parâmetro e não retorna nenhum valor (por isso é do tipo **void**).
6. Dentro da função **imprimirVariaveisGlobais()**, os valores das variáveis globais **var1** e **var2** são impressos usando **printf()**.
7. A função **main()** é o ponto de entrada do programa.
8. Na função **main()**, a função **imprimirVariaveisGlobais()** é chamada para imprimir o valor inicial das variáveis globais, que neste caso são 0.
9. Em seguida, a função **modificarVariaveisGlobais()** é chamada para modificar o valor das variáveis globais para 10 e 20, respectivamente.
10. Após a modificação, a função **imprimirVariaveisGlobais()** é chamada novamente para imprimir o novo valor das variáveis globais.

20. Crie uma função que declare uma variável estática e outra função que declare uma variável automática (não-estática). Ambas as funções devem incrementar o valor das respectivas variáveis e imprimir o seu valor. Chame estas funções várias vezes no seu programa principal e observe o comportamento das variáveis estática e automática.

```c
#include <stdio.h>

// Função 'incrementarStatic' com variável estática
void incrementarStatic(){
    // Variável estática 'num' que mantém seu valor entre chamadas
    static int num = 0;
    // Incrementa o valor da variável estática 'num'
    num++;
    // Imprime o valor atual da variável estática
    printf("Valor atual da variavel estatica: %d\n", num);
}

// Função 'incrementarAutomatica' com variável automática
void incrementarAutomatica() {
    // Variável automática 'num' que é recriada a cada chamada da função
    int num = 0;
    // Incrementa o valor da variável automática 'num'
    num++;
    // Imprime o valor atual da variável automática
    printf("Valor atual da variavel automatica: %d\n", num);
}

int main() {
    // Chama a função 'incrementarStatic' três vezes
    incrementarStatic();
    incrementarStatic();
    incrementarStatic();

    // Chama a função 'incrementarAutomatica' três vezes
    incrementarAutomatica();
```

```
31.     incrementarAutomatica();
32.     incrementarAutomatica();
33.
34.     return 0;
35. }
```

Explicação do código:

1. A biblioteca **stdio.h** é incluída para permitir o uso de funções de entrada/saída padrão em C.
2. A função **incrementarStatic()** é definida abaixo da função **main()**. Essa função não recebe nenhum parâmetro e não retorna nenhum valor (por isso é do tipo **void**).
3. Dentro da função **incrementarStatic()**, é declarada uma variável estática **num**. A variável estática mantém seu valor entre as chamadas da função, ou seja, ela não é reinicializada a cada vez que a função é chamada.
4. Em cada chamada da função **incrementarStatic()**, o valor da variável estática **num** é incrementado em 1.
5. Em cada chamada da função **incrementarStatic()**, é impresso o valor atual da variável estática **num** usando **printf()**.
6. A função **incrementarAutomatica()** é definida abaixo da função **main()**. Essa função não recebe nenhum parâmetro e não retorna nenhum valor (por isso é do tipo **void**).
7. Dentro da função **incrementarAutomatica()**, é declarada uma variável automática **num**. A variável automática é recriada a cada chamada da função e não mantém seu valor entre as chamadas.
8. Em cada chamada da função **incrementarAutomatica()**, o valor da variável automática **num** é incrementado em 1.
9. Em cada chamada da função **incrementarAutomatica()**, é impresso o valor atual da variável automática **num** usando **printf()**.
10. A função **main()** é o ponto de entrada do programa.
11. Na função **main()**, a função **incrementarStatic()** é chamada três vezes.
12. Em seguida, a função **incrementarAutomatica()** é chamada três vezes.