Universidade Federal do Pará – UFPA

Instituto de Tecnologia – ITEC

Programa de Educação Tutorial de Engenharia Elétrica – PET-EE

MINICURSO DE PROGRAMAÇÃO EM C

2021

Sumário

1. AMBIENTE DEV-C

A linguagem de programação C possui a programação estruturada como paradigma de programação, isto é, o padrão da linguagem consiste em sequência, decisão e iteração. A sequência implica que o código é executado de forma sequencial, uma tarefa seguida da outra, como se cada linha fosse lida uma única vez. Já na decisão são introduzidos testes lógicos, as tarefas são executadas a partir de um teste lógico, verdadeiro ou falso, 1 ou 0. E por fim, na iteração, trechos do código são repetidos um certo número finito de vezes sob um teste lógico.

Ao se utilizar um compilador para escrever seus códigos sempre tenha atenção aos avisos e erros que ele apontar, 'debugar' um código pode ser uma tarefa árdua, por isso toda ajuda é bem-vinda, erros com nomes de funções e⁄ou, ausência de ponto e vírgula (;), comandos errados são exemplos de erros que o compilador aponta, basta prestar atenção nos alertas dados.

A Figura 1 mostra o ambiente inicial do software Dev-C, que é um programa computacional que permite a construção de códigos em linguagem C.

Para começar a elaboração de um novo script, vá para Arquivo→Novo→Arquivo Fonte.

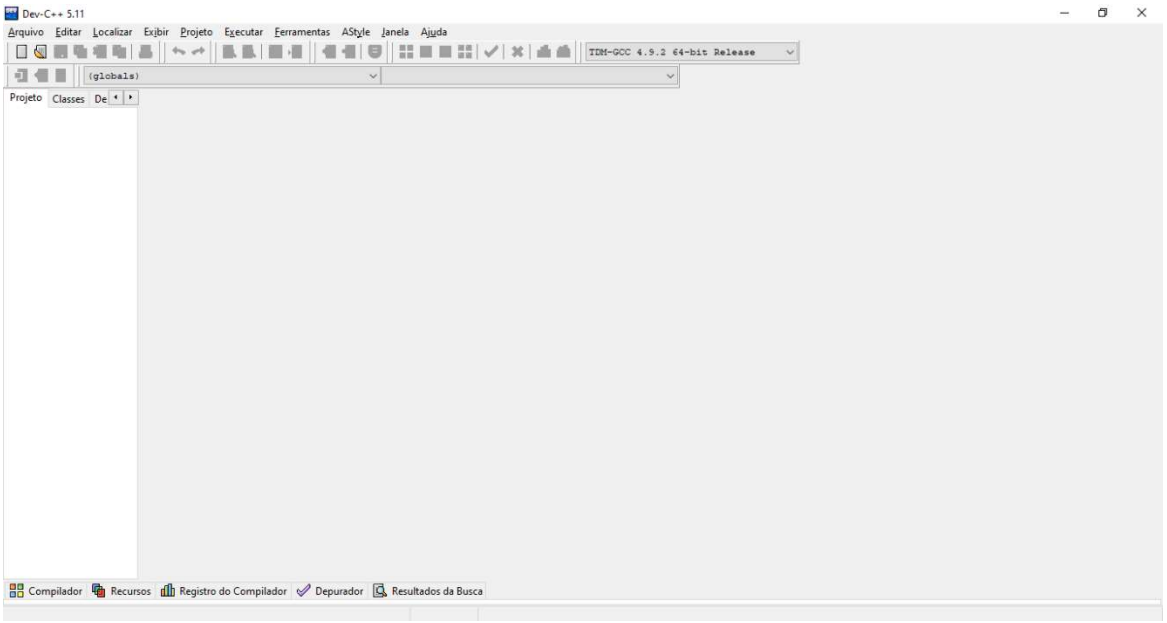

Figura 1. Ambiente Dev C

A linguagem de programação C relaciona-se com uma metodologia de tornar a programação algo mais acessível as pessoas, e mais distante da linguagem de máquina, que trabalha com códigos binários (1 e 0). Os dois pontos fundamentais são a inserção de bibliotecas e as funções do programa.

Exemplo 01: Uma pessoa é obesa se seu índice de massa corpórea é superior a 30, tal índice é a razão entre seu peso e o quadrado da sua altura.

```
/* Programa que informa se uma pessoa é ou não    obesa */

#include<stdio.h>

#include<math.h>
```

```
#define LIMITE 30

main(){

float peso, altura, imc;

printf("\n Digite seu peso e sua altura: ");
scanf("%f" "%f", &peso, &altura);

imc = peso/pow(altura,2);
printf("\n Seu imc é: %.1f", imc);

if(imc<=LIMITE){
   printf("\n Você não está obeso(a).");
}

else{
   printf("\n Você está obeso(a)");
}

system("pause");
return 0;

}
```

Perceba que os acentos da língua portuguesa não foram aceitos, para tal, deve-se acrescentar as seguintes linhas:

```
#include<locale.h>

main(){
setlocale(LC_ALL, “Portuguese”);
}
```

Como esse programa é muito simples, ele consiste de uma única função: main. Essa

função solicita os dados da pessoa, calcula o seu índice de massa corpórea e informa se ela está obesa ou não.

Alguns pontos desse primeiro exemplo devem ser ressaltados:

- Todo texto delimitado por /* e */ é considerado como comentário, isto é, serve apenas para esclarecer algum ponto específico do programa;
- A diretiva #include causa a inclusão de arquivos de cabeçalho contendo declarações necessárias à compilação. Os arquivos stdio.h e math.h declaram, respectivamente, comandos de E/S padrão e funções matemáticas. A diretiva #define declara constantes simbólicas;
- Os parênteses após o nome de uma função, como em main(), são obriga- tórios. Além disso, o compilador distingue maiúsculas e minúsculas e, portanto, o nome main é reservado, mas Main não o é;
- As chaves { e } servem para delimitar um bloco de instruções. As variáveis devem ser declaradas antes de serem usadas, logo no início do bloco;
- As funções scanf() e printf() realizam entrada e saída de dados padrão;
- Cálculos e comparações são efetuados com os operadores aritméticos, funções matemáticas e operadores relacionais convencionais. A atribuição de valores às variáveis é realizada pelo operador =.

2. TIPOS DE DADOS

Programas servem essencialmente para manipular dados: ao executarmos um programa, fornecemos-lhes dados e esperamos que ele faça alguma coisa com eles. Os dados podem se apresentar em duas formas distintas: como constantes ou variáveis. No programa do exemplo 1, o número 30 é um dado constante, enquanto o peso e a altura da pessoa são dados variáveis, isto é, são dados cujos valores variam de uma execução para outra. Além da distinção entre as formas em que os dados podem se apresentar, existe também uma distinção entre os tipos de dados que o computador é capaz de manipular; como, por exemplo, números, letras, palavras, etc. A linguagem C oferece cinco tipos de dados básicos, como pode ser visto no quadro 1.

Quadro 1 – Tipos de dados da Programação em C.

Tipo	Espaço	Escala
Char	1 byte	-128 a +127
Int	2 bytes	-32768 a +32767
Float	4 bytes	3.4e-38 a 3.4e+38
Double	8 bytes	1.7e-308 a 1.7e+308
Void	nenhum	nenhuma

Em C não existe muita distinção entre a representação gráfica de um caractere e o seu código ASCII. Como sabemos, o computador somente é capaz de manipular números. Então, quando atribuímos um caractere a uma variável, estamos na verdade armazenando o seu código ASCII (que na tabela padrão varia de 0 a 127). É por este motivo que a escala de valores do tipo char varia no intervalo de –128 a +127. Para representar um caractere constante, basta escrevê-lo entre apóstrofos como, por exemplo, 'A'.

As variáveis do tipo *int* podem armazenar números maiores que as variáveis do tipo char, mas também gastam mais espaço de memória. Valores fracionários podem ser armazenados em variáveis do tipo *float* ou *double*, conforme a necessidade. Já o tipo *void* é um tipo especial, que tem aplicação mais avançada, e seu uso será visto mais adiante.

A declaração de uma variável consiste em um tipo e um identificador. O tipo determina o espaço de memória que deverá ser alocado para ela e o identificador permitirá que ela seja referenciada no restante do programa.

3. ENTRADA/SAÍDA DE DADOS

A função scanf() permite que um valor seja lido do teclado e armazenado numa variável. Sua sintaxe consiste numa cadeia de formatação seguida de uma lista de argumentos, cada um deles sendo o endereço de uma variável:

scanf("formatação", arg1, arg2, ..., argn);

A cadeia de formatação é composta por códigos especiais, denominados especificadores de formato, que indicam a quantidade e os tipos dos dados que serão lidos pela função.

Exemplo 2: Lendo dados com a função scanf().

```
#include<stdio.h>
main(){
int idade;
printf("Digite sua idade: ");
scanf("%d", &idade);
system("pause");
return 0;
}
```

Como podemos observar, a cada especificação de formato na cadeia de formatação corresponde um endereço de variável na lista de argumentos. Existem alguns especificadores de acordo com o tipo de dado a ser analisado, mostrados no quadro 2.

Quadro 2 – Alguns especificadores da Linguagem C

Especificador	Representa
%c	um único caractere
%o, %d, %x	um número inteiro em octal, decimal ou hexadecimal
%u	um número inteiro em base decimal sem sinal
%ld	um número inteiro longo em base decimal
%f, %lf	um número real de precisão simples ou dupla
%s	uma cadeia de caracteres (string)
%%	um único sinal de porcentagem

A função printf( ) nos permite exibir informações formatadas no vídeo. A sua sintaxe é essencialmente idêntica àquela da função scanf(). A principal diferença é que agora a lista de argumentos deve conter os valores a serem exibidos e não seus endereços:

printf("formatação", arg1, arg2, ..., argn);

Além disso, a cadeia de formatação pode conter também texto, que é exibido normalmente, e caracteres de controle, cuja exibição causa efeitos especiais, quadro 3.

Quadro 3 – Principais caracteres de controle

Caracteres	Efeitos
\a	soa o alarme do microcomputador
\b	o cursor retrocede uma coluna
\f	alimenta página na impressora
\n	o cursor avança para uma nova linha
\r	o cursor retrocede para a primeira coluna da linha
\t	o cursor avança para próxima marca de tabulação
\"	exibe uma única aspa
\'	exibe um único apóstrofo
\\	exibe uma única barra invertida

Desses caracteres, o mais usado é '\n'. Através dele podemos indicar quando uma nova linha deve ser utilizada ao se exibir alguma informação na tela.

Exemplo 3: Exibindo dados com a função printf().

```
#include <stdio.h>
#define PI 3.1415
```

```
main() {
double raio, perim;
printf("\n Qual a medida do raio?");
scanf("%lf", &raio);
perim = 2 PI raio;
printf("\n O perímetro da circunferência é %lf", perim);
system(“pause”);
return 0;
}
```

4. OPERADORES ARITMÉTICOS

A linguagem C oferece operadores para as quatro operações aritméticas e também um operador para calcular o resto da divisão entre dois números inteiros, quadro 4.

Quadro 4 – Operadores aritméticos em C.

Operador	Resultado
+	soma de dois números quaisquer
-	diferença entre dois números quaisquer
*	produto de dois números quaisquer
/	quociente da divisão de dois números
%	resto da divisão de dois números inteiros

Destes operadores, apenas dois merecem atenção especial. Os demais funcionam conforme as regras usuais estabelecidas na matemática básica.

- Divisão: o operador de divisão fornece resultado inteiro apenas quando ambos os operandos são inteiros. Por exemplo, 7 / 2 ⇒ 3 e 7.0 / 2 ⇒ 3.5.
- Resto: o operador de resto somente pode ser utilizado com operandos inteiros. Por exemplo, 7 % 2 ⇒ 1 e 7.0 % 2 ⇒ erro.

5. OPERADORES LÓGICOS

Em C, não existe um tipo específico para a representação de valores lógicos. Entretanto, qualquer valor pode ser interpretado como um valor lógico: “zero representa falso e qualquer outro valor representa verdade”. Por exemplo, os valores 5, –3, 1.2 e 'a' são verdadeiros, enquanto 0 e 4–4 são falsos.

Para gerar um valor lógico, usamos os operadores relacionais, quadro 5. Através deles podemos comparar dois valores de diversas formas. O resultado da avaliação de um operador relacional é 0 se a comparação é falsa e 1 se verdadeira.

Quadro 5 – Operadores relacionais em C.

Operador relacional	Resultado
x == y	verdade se x for igual a y
x != y	verdade se x for diferente de y
x < y	verdade se x for menor que y
x > y	verdade se x for maior que y
x <= y	verdade se x for menor ou igual a y
x >= y	verdade se x for maior ou igual a y

Além dos operadores relacionais, C oferece também operadores lógicos, quadro 6. Com eles, podemos criar expressões lógicas compostas. Os operadores lógicos funcionam conforme as regras definidas na lógica matemática.

Quadro 6 – Operadores lógicos em C

Operador lógico	Resultado
! x	verdade se e só se x for falso
x && y	verdade se e só se x e y forem verdade
x || y	verdade se e só se x ou y for verdade

6. COMANDOS CONDICIONAIS

A estrutura condicional ou de decisão simples serve para escolher um entre dois comandos alternativos. Em C, a estrutura condicional é codificada da seguinte forma:

if(condição) comando1; else comando2;

Exemplo 4: O uso de decisão simples.

```
#include <stdio.h>
main() {
float a, b, m;
printf("\n Informe as duas notas obtidas: ");
scanf("%f %f", &a, &b);
m = (a+b)/2;
if( m >= 7.0 ) printf("\n Aprovado"); else printf("\n Reprovado");
system("pause";
return 0;
}
```

É possível que, algumas vezes, um destes comandos alternativos (ou ambos) sejam também condicionais. Nesse caso, dizemos que o primeiro condicional é o principal e o outro está aninhado ou encadeado, conforme indicado a seguir:

```
if( condição )      /* principal  */
if ...      /* aninhado */
else
if ...      /* encadeado */
```

Para exemplificar o uso desses condicionais, vamos considerar o seguinte problema: "Dados três números verificar se eles podem representar as medidas dos lados de um triângulo e, se puderem, classificar o triângulo em equilátero, isósceles ou escaleno".

Exemplo 5: O uso de condicionais aninhados e encadeados.

```
#include <stdio.h>
main() {
```

```
float a, b, c;
printf("\nInforme três números: "); scanf("%f %f %f", &a, &b, &c);
if( a<b+c && b<a+c && c<a+b ) {
printf("\n É um triângulo: ");
if( a==b && b==c ) {
printf("equilátero");
}
else if( a==b || a==c || b==c ){
 printf("isósceles");
}
else {
printf("escaleno");
}
}
else {
printf("\n Não é um triângulo");
}
}
```

A estrutura de decisão múltipla é bastante adequada quando precisamos escolher uma entre várias alternativas previamente e definidas, por exemplo, num menu. A decisão múltipla tem a seguinte forma básica:

```
switch( expressão ) {
case constante1 : comando1; break;
case constante2 : comando2; break;
...
case constanten : comandon; break;
default   : comando;
}
```

Note que, embora o comando break seja quase sempre usado juntamente com o comando switch, ele não faz parte da sintaxe desse comando. Se dois casos não são separados por um comando break, dizemos que o controle "vaza" de um caso para o outro, ou seja, quando o primeiro caso é selecionado para execução, não apenas o comando associado a ele é executado, mas também o comando associado ao segundo.

Exemplo 6: O uso da estrutura de decisão múltipla com vazamentos.

```
#include <stdio.h>
main() {
int n;
printf(“\n Digite um número: ”);
scanf(“%d”, &n);
switch( n ) {
case 1: printf(“A”);
break;
case 2: printf(“B”);
break;
case 3: printf(“C”);
case 4: printf(“D”);
break;
default: printf(“Z”);
}
system(“pause”);
return 0;
}
```

Exemplo 7: Decisão múltipla e finalização do programa.

```
#include<stdio.h>
#include<locale.h>
main(){
setlocale(LC_ALL,"Portuguese");
int materia;
printf("Têm-se as seguintes matérias: \n");
printf("1. Matemática \n");
printf("2. Física \n");
printf("3. Química \n");
printf("Digite o número da sua matéria favorita ou qualquer tecla para abandonar o programa: ");
scanf("%d",&materia);
```

```
switch(materia){

case 1: printf("Sua matéria favorita é matemática. \n");
break;
case 2: printf("Sua matéria favorita é física. \n");
break;

case 3: printf("Sua matéria favorita é química. \n");
break;
default: return 0;
}
system("pause");
return 0;
}
```

7. LAÇOS DE REPETIÇÃO

A estrutura de repetição com contador tem seu funcionamento controlado por uma variável que conta o número de vezes que o comando é executado. Em C, essa estrutura é implementada pelo comando for, cuja forma básica é a seguinte:

for (inicialização; condição; alteração) comando;

A inicialização é uma expressão que atribui um valor inicial ao contador, a condição verifica se a contagem chegou ao fim e a alteração modifica o valor do contador. Enquanto a contagem não termina, o comando associado ao for é repetidamente executado.

Exemplo 8: Uma contagem progressiva.

```
#include <stdio.h>
main() {
int c;
for(c=1; c<=9; c++) {
printf(“%d ”, c);
}
system(“pause”);
```

```
return 0;
}
```

A saída produzida pelo código será: 1 2 3 4 5 6 7 8 9.

A estrutura de repetição com precondição é mais genérica que aquela com contador. Em C, ela tem a seguinte forma:

```
while (condição) comando;
```

Seu funcionamento é controlado por uma única expressão, sua condição, cujo valor deve ser verdadeiro para que o comando seja repetido. A repetição com precondição para somente quando sua condição se torna falsa.

Para exemplificar o uso de repetição com precondição, vamos resolver o seguinte problema: "dado um número natural, exibir os seus dígitos". Por exemplo, dado o número 8503 como entrada, o programa deverá exibir como saída os dígitos 3, 0, 5 e 8. A estratégia será dividir o número sucessivamente por 10 e ir exibindo os restos obtidos, um a um. O processo se repete enquanto o número for diferente de zero.

Exemplo 9: Exibe os dígitos de um número.

```
#include <stdio.h>
main() {
int n, d;
printf("\n Digite um número: ");
scanf("%d", &n);
printf("\n Os seus dígitos são: ");
while( n != 0 ) {
d = n % 10;
n /= 10;
printf("%d ", d);
}
system("pause");
return 0;
```

8. VETORES E MATRIZES

A necessidade de se trabalhar com vetores e matrizes surge a partir de problemas que

precisam armazenar uma grande quantidade de valores como por exemplo as notas de uma turma, nesse caso torna-se mais viável criar uma única variável capaz de armazenar as notas de todos os estudantes em vez de criar uma variável para cada nota de aluno.

A forma geral para declarar uma unidimensional é:

```
tipo nome_variavel [tamanho];
```

É necessário especificar o 'tipo' da matriz, para que cada elemento tenha uma definição explícita de qual é seu tipo de dado, 'nome_variavel' é apenas um nome escolhido pelo programador, e 'tamanho' faz referência a quantidade de elementos da matriz. Por exemplo, para criar uma matriz com dados inteiros, chamada 'valores' e com 15 elementos, faz-se:

```
int valores[15];
```

Um ponto importante é que uma matriz sempre inicia com o índice '0', ou seja, ao declarar float notas[10], você está criando uma matriz unidimensional de 10 elementos, de notas[0] até notas[9].

Exemplo 10: Mostrando valores de um vetor

```
#include <stdio.h>
int main()
{
   int notas[3]={0};
   printf("%i\n", notas[0]);
   printf("%i\n", notas[1]);
   printf("%i\n", notas[2]);
   system("pause");
   return 0;
}
```

Exemplo 11: Mostrando valores de um vetor com Ciclo For

```
#include<stdio.h>
int main(){
        int vetor[3] = {0};
        int i;
        for(i = 0; i<3; i++){
```

```
	printf("%d\n", vetor[i]);
}
	return 0;
}
```

Exemplo 12: Calculando média das avaliações com vetores

```
#include <stdio.h>
#include <math.h>
#include<locale.h>

main(){

setlocale(LC_ALL,"Portuguese");

float notas[3];
	float soma=0;
	float media;
	int c;

	for(c=0; c<3; c++){
	printf("Digite a nota da %da avaliação:",c+1);
	scanf("%f", &notas[c]);
	soma = soma + notas[c];
}

	media = soma/3;
	printf("A media e: %f", media);
system("pause");
return 0;
}
```

Exemplo 13: Mostrando valores de uma matriz

```
#include<stdio.h>
int main(){
	int matriz[2][2] = {1,2,3,4};
	printf("%i", matriz[0][0]);
```

```
	printf("%i", matriz[0][1]);
	printf("%i", matriz[1][0]);
	printf("%i", matriz[1][1]);
	return 0;
}
```

Exemplo 14: Mostrando valores de uma matriz com Ciclo For

```
#include<stdio.h>
int main(){
	int matriz[3][3] = {1,2,3,4,5,6,7,8,9};
	int m,n;
	for(m=0; m<3; m++){
		printf("\n");
		for(n=0; n<3; n++){
	printf("%d",matriz[m][n]);
}
}
	return 0;
}
```

Exemplo 15: Calculando determinante de uma matriz 2x2

```
#include<stdio.h>
int main(){
	int matriz[2][2];
	int m, n, det;
	for(m=0; m<2; m++){
		printf("\n");
		for(n=0; n<2; n++){
			printf("Digite o valor da posicao %d %d:\n", m, n);
			scanf("%d", &matriz[m][n]);
		}
	}
	det=(matriz[0][0]*matriz[1][1])-(matriz[0][1]*matriz[1][0]);
	printf("O determinante e: %d", det);
	return 0;
}
```

9. FUNÇÕES

Funções são blocos do programa nos quais uma determinada atividade ocorre. A forma geral de uma função é:

```
Especificador_de_tipo nome_função (lista_parâmetros){
corpo da função
}
```

O especificador determina qual o tipo de dado que a função deve retornar ao se usar o comando “return”. Se nenhum tipo for definido, o compilador entende que deve retornar um valor inteiro. Lista_parâmetros equivale a uma lista com nomes de variáveis e seus tipos, separados por vírgulas, as quais representam os argumentos de entrada que a função precisa para funcionar adequadamente. Uma função pode não ter parâmetros, mesmo assim são necessários os parênteses ().

Obs: A linguagem C não é tecnicamente uma linguagem estruturada em blocos, por isso não é possível chamar uma função dentro de uma função.

Exemplo 16: Calcular o quadrado de um número utilizando uma função.

```
//Calcular o quadrado de um número
#include<stdio.h>
#include<locale.h>
main(){
setlocale(LC_ALL,"Portuguese");
int numero;
printf("Digite um valor: ");
scanf("%d",&numero);
printf("O quadrado do número %d é: %d \n",numero,quadrado(numero));

system("pause");
return 0;
}
int quadrado(int x){
x=x*x;
```

```
return x;
}
```

O comando return tem dois usos, ele pode ser aplicado para saída da função em que os comandos estavam sendo executados, ou para retornar um determinado valor.

Exemplo 17: Calcular área do quadrado a partir da diagonal usando função.

```
//Calcular área do quadrado

#include<stdio.h>
#include<locale.h>
#include<math.h>

float calcular_area(float diagonal);

main(){

setlocale(LC_ALL,"Portuguese");

float diagonal,area;

printf("Digite a diagonal do quadrado: ");
scanf("%f",&diagonal);

area=calcular_area(diagonal);

printf("A área é: %f \n",area);
```

```
system("pause");
return 0;
}

float calcular_area(float x_diagonal){

float lado,area_real;

lado = x_diagonal/pow(2,0.5);

area_real = pow(lado,2);

return area_real;
}
```

Exemplo 18: Usando função a partir de comando switch.

```
#include<stdio.h>
#include<locale.h>

main(){

setlocale(LC_ALL,"Portuguese");

int serie;

printf("1. La casa de papel. \n");
printf("2. The Witcher. \n");
printf("3. The walking dead. \n");

printf("Digite o número da sua série favorita: \n");

do{
   scanf("%d",&serie);
```

```
    switch(serie){

    case 1:  casa_papel();
    break;

    case 2: witcher();
    break;

    case 3: dead();
    break;

    }
} while(serie!=1 && serie!=2 && serie!=3);

system("pause");
return 0;
}

casa_papel(){

printf("Sua série favorita é La casa de papel. \n");

}

witcher(){

printf("Sua série favorita é The witcher. \n");

}

dead(){
```

```
printf("Sua série favorita é The walking dead. \n");

}
```

Exemplo 19: Criando um vetor que é igual ao dobro do preenchido pelo usuário.

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<math.h>
#include<locale.h>

int main(){

int vetor[3],vetorA[3],i;

int dobro(int x){
return(2*x);
}

for(i=0;i<3;i++){
   printf("Informe o %do numero: ",i+1);
   scanf("%d",&vetor[i]);
}
printf("\n");
printf("Vetor:[");

for(i=0;i<3;i++){
   printf("%d",vetor[i]);
```

```
    if(i<2){
        printf(",");
    }
}
printf("]");
printf("\n");

for(i=0;i<3;i++){
    vetorA[i]=dobro(vetor[i]);
}

printf("\n");
printf("Vetor dobro:[");

for(i=0;i<3;i++){
    printf("%d",vetorA[i]);
    if(i<2){
        printf(",");
    }
}
printf("]");
printf("\n");

system("pause");
return 0;
}
```

Exemplo 20: Retorna o maior valor entre 2 números.

```
#include <stdio.h>
float maior2( float num1, float num2);
```

```
void main(void)
{
   float x,y,maiorvalor;

   printf("digite dois valores a serem comparados \n");
   scanf("%f %f",&x,&y);

   maiorvalor = maior2(x,y);

   printf("o maior e %.1f ", maiorvalor);
}

float maior2( float num1 , float num2)
{
   if(num1>=num2)
      return num1;
   else
      return num2;
}
```

Exemplo 21: Cálculo do fatorial de um número.

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

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<math.h>
#include<locale.h>

int fatorial(int x){
   int fator,i;
   fator=x;
for(i=1;i<x;i++){
   fator=fator*(x-i);
}

return fator;
}

int main(){

int numero,a;

setlocale(LC_ALL,"PORTUGUESE");

printf("Digite um número: ");
scanf("%d",&numero);

a=fatorial(numero);

printf("O fatorial de %d é %d. \n",numero,a);
```

```
system("pause");

return 0;

}
```

10.EXERCÍCIOS:

1) Dadas as duas notas de um aluno, informe a sua média final.

2) Dada uma temperatura em graus Fahrenheit, informe o valor correspondente em graus Celsius. [Dica: C = (F – 32) * (5 / 9)].

3) Dados dois números distintos, informe qual dele é o maior.

4) Uma empresa determinou um reajuste salarial de 5% a todos os seus funcionários. Além disto, concedeu um abono de R$ 100,00 para aqueles que recebem até R$ 750,00. Dado o valor do salário de um funcionário, informar para quanto ele será reajustado.

5) Dados os coeficientes (a≠0, b e c) de uma equação do 2º grau, calcule e informe suas raízes reais, usando a fórmula de Báskara.

6) Dados a altura e o sexo de uma pessoa, determine seu peso ideal de acordo com as fórmulas a seguir:
   *para homens o peso ideal é 72.7*altura - 58
   *para mulheres o peso ideal é 62.1*altura - 44.7

7) Dado um valor n, exiba uma contagem regressiva.

8) Dados um número natural n, exiba seu fatorial n!

9) O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, calcule seu quadrado usando a soma de ímpares ao invés de produto.

10) Um número natural é triangular se é igual à soma dos n primeiros números naturais consecutivos, a partir de 1. Por exemplo, 1, 3, 6, 10, 15, ... são triangulares. Dado um natural $n \geq 1$, informe se ele é triangular.

11.DESAFIOS:

01) Crie um programa que pergunte qual o nome do usuário e retorne "Seja bem vindo (a) nome do usuário".

02) O usuário deve inserir um número no formato centena-dezena-unidade e o programa deve responder do seguinte modo: "Centena = dígito das centenas, dezena = dígito das dezenas, unidade = dígito das unidades".

03) Faça um programa que receba um valor inteiro, um valor decimal e uma letra do alfabeto. Em seguida, imprima o tipo de dado que cada um corresponde.

04) Pergunte ao usuário qual o dia da semana (1 até 7) e retorne o nome do dia (domingo até sábado).

05) Quando o usuário digitar um número, retorne o quadrado desse número. Obs: preserve o sinal (10 ao quadrado é 100; -10 ao quadrado é -100);

06) Preencha um vetor com 10 elementos, e retorne os elementos na ordem inversa.

Ex: vetor de entrada [1,2,3,4,5,6,7,8,9,10] → vetor de saída [10,9,8,7,6,5,4,3,2,1]

07) Preencha 2 vetores de 5 elementos e retorne um terceiro vetor somando cada elemento.

Ex: vetorA [1,2,3,4,5] vetorB [1,2,3,4,5] → vetor3 [2,4,6,8,10]