119.Escreva uma função que retorne o k-ésimo dígito (da direita para esquerda) de um

inteiro n, k e n dados. Por exemplo, K\_esimoDigito(2845, 3) = 8.

#include <stdio.h>

int k\_esimoDigito(int n, int k)

{

int d;

if(k==1){

return n%10;

}

else{

d=n/10;

k\_esimoDigito(d,k-1);

}

}

int main(void)

{

int digito,n,k;

printf("Digite um inteiro (n) ");

scanf("%d",&n);

printf("qual posição ? ");

scanf("%d",&k);

digito=k\_esimoDigito(n,k);

printf("Digito %d : %d",k,digito);

return 0;

}

120.O fatorial ímpar de um número n ímpar positivo é o produto de todos os números

ímpares positivos menores do que ou iguais a n.Indicando o fatorial ímpar de n por

n| temos, n|=1.3.5.....n. Por exemplo, 7| = 1 . 3 . 5 . 7 = 105. Escreva funções

iterativas e recursivas para a determinação do fatorial ímpar de um inteiro ímpar

dado.

#include <studio.h>

int fatorial\_recursivo(int n)

{

if ((n==1)||(n==3)){

return n;

}

else{

return n\*fatorial\_recursivo(n-2);

}

}

int fatorial\_iterativo(int n)

{

int i=1,fat=1;

while(i<=n){

fat=fat\*i;

i+i+2;

}

return fat;

}

int main(void)

{

int fat\_it,fat\_rec,n;

printf("digite n: ");

scanf("%d",&n);

fat\_it=fatorial\_iterativo(n);

fat\_rec=fatorial\_recursivo(n);

printf("fat\_it(%d): %d",n,fat\_it);

printf("fat\_rec(%d); %d",n,fat\_rec);

return 0;

}

121.Como na questão anterior, o fatorial primo (ou primorial) de um número primo

positivo é o produto de todos os primos positivos menores do que ou iguais a ele:

p# = 2 . 3 . 5 . 7 . ... .p (sendo 2# = 2). Por exemplo, 7# = 2 . 3 . 5 . 7 = 210. Escreva

um programa que determine o fatorial primo de um primo dado.

#include <stdio.h>

int teste\_eh\_primo(int n)

{

int i;

int eh\_primo=1;

int k=n/2;

if (n==1){

eh\_primo=0;

}

else{

for(i=2;i<=k;i++){

if((n%i)==0){

eh\_primo=0;

break;

}

}

return eh\_primo;

}

}

int fatorial\_primo(int n)

{

int verifica\_primo;

int k;

if(n==2){

return 2;

}

else{

verifica\_primo=teste\_eh\_primo(n);

/\* se for primo eu continuo a multiplicação pelo proximo elemento primo\*/

if(verifica\_primo){

return n\*fatorial\_primo(n-1);

}

else{

k=n-1;

return fatorial\_primo(k);

}

}

}

int main(void)

{

int n;

int eh\_primo;

int nat;

printf("\nfatorial\_primo de : ");

scanf("%d",&n);

nat=fatorial\_primo(n);

printf("\nfatorial de %d: %d",n,nat);

return 0;

}

122.Escreva funções, iterativa e recursiva, que retornem a soma dos algarismos de um

inteiro positivo dado.

123.Escreva uma função recursiva que retorne o n-ésimo termo da sequência de

Fibbonaci, n dado.

#include <studio.h>

int fibonacci(int n);

{

if ((n==1)||(n==0)){

return n;

}

else{

return fibonacci(n-1)+fibonacci(n-2);

}

}

int main(void)

{

int n,fib;

printf("Digite n:");

scanf("%d",&n);

fib=fibonacci(n);

printf("fibo(%d): %d",n,fib);

return 0;

}

124.Escreva uma função, interativa e recursiva, que retorne o fatorial de um número.

#include <stdio.h>

int fatorial(int n)

{

if ((n==0)||(n==1)){

return n;

}

else{

return n\*fatorial(n-1);

}

}

int main(void)

{

int n, fat;

printf("fatorial de: ");

scanf("%d",&n);

fat=fatorial(n);

printf("fatorial de %d é %d",n,fat);

return 0;

}

125.

#include<stdio.h>

void gotox(int x) {

int i;

for (i=0; i < x; i++) printf("%c",32);

}

int main() {

int i, j, n = 9;

for (i = 1; i < n; i++) {

for (j = 1; j < n; j++) {

printf("%d\*%d = %d", i, j, i\*j);

if (i\*j < 10) gotox(3);

else gotox(2);

}

printf("\n\n");

}

return 0;

}

126.

#include<stdio.h>

// mmc é feito dividindo ambos os numeros pelo seu divisor comum,

// por isso é necessario o mdc.

int mdc(int a, int b) {

if(b == 0) return a;

return mdc(b, a%b);

}

int mmc(int a, int b) {

int div;

if(b == 0) return a;

div = (a\*b)/(mdc(a, b));

return div;

}

int main() {

printf("mmc de %d e %d: %d", 12, 45, mmc(12, 45));

return 0;

}

127.

#include<stdio.h>

// mmc é feito dividindo ambos os numeros pelo seu divisor comum,

// por isso é necessario o mdc.

int ppow(int num, int exp) {

if(exp == 1) return num;

return num \* ppow(num, exp-1);

}

int main() {

printf("%d elevado a %d: %d", 2, 5, ppow(2,5));

return 0;

}

128.129.130

#include<stdio.h>

// mmc é feito dividindo ambos os numeros pelo seu divisor comum,

// por isso é necessario o mdc.

int recMax(int \*vet, int size\_of, int max) {

if (max < vet[size\_of]) max = vet[size\_of];

if (size\_of == 1) return max;

return recMax(vet, size\_of-1, max);

}

int max(int vet[], int len) {

return recMax(vet, len, -99999);

}

void showVet(int \*vet, int len) {

int i;

for (i = 0; i < len; i++) printf("%d, ", vet[i]);

}

int main() {

int vet[] = {1,5,3,8,2,10,7,6,4};

int len = sizeof(vet)/sizeof(int) - 1;

printf("o maior valor do vetor: ");

showVet(vet, len);

printf("é: %d", max(vet, len));

return 0;

}

131.

#include<stdio.h>

int f(int n) {

if (n < 4) return 3 \* n;

return (2 \* f(n - 4)) + 5;

}

int main() {

printf("f(3): %d", f(3));

printf("\n");

printf("f(7): %d", f(7));

return 0;

}

132.

#include<stdio.h>

int modulo(int value) {

if (value < 0) return value \* -1;

return value;

}

int ppow(int num, int exp) {

if(exp == 1) return num;

return num \* ppow(num, exp-1);

}

int raizq(int x, int x0, int e) {

if (modulo(ppow(x0, 2) - x) <= e) return x0;

return raizq(x, (ppow(x0, 2) + x)/(2\*x0), e);

}

int main() {

printf("raizq(13, 3.2, 0.001): %d", raizq(13, 3.2, 0.001));

return 0;

}

133.

#include<stdio.h>

int ark(int m, int n) {

if (m == 0) return (n + 1);

else if (m > 0 && n == 0) return ark(m - 1, 1);

else if (m > 0 && n > 0) return ark(m - 1, ark(m, n - 1));

printf("valor negativo nao eh valido");

return -1;

}

int main() {

printf("a(1, 2): %d", ark(1, 2));

return 0;

}

134.

#include<stdio.h>

int modulo(int value) {

if (value < 0) return value \* -1;

return value;

}

int ppow(int num, int exp) {

if(exp == 1) return num;

return num \* ppow(num, exp-1);

}

int f(int x, int n) {

// expoente negativo

if (n < 0) return 1 / f(x, modulo(n));

// casos base

if (n == 0) return 1;

else if (n == 1) return x;

else if (n == 2) return ppow(x, 2);

// recursiva

if (n%2 == 1) return x \* f(x, n - 1);

else return ppow(f(x, n/2), 2);

}

int main() {

printf("f(2, 8): %d", f(3, 5));

return 0;

}